Fall 1-10-2020

Investigating Patterns in Convolution Neural Network Parameters Using Probabilistic Support Vector Machines

Yuqiu Zhang
Washington University in St. Louis

Washington University in St. Louis

McKelvey School of Engineering

Department of Electrical and Systems Engineering

Thesis Examination Committee:
Shantanu Chakrabartty, Chair
Ayan Chakrabarti
Arye Nehorai

Investigating Patterns in Convolution Neural Network Parameters Using Probabilistic Support Vector Machines

by

Yuqiu Zhang

A thesis presented to the McKelvey School of Engineering
of Washington University in partial fulfillment of the
requirements for the degree of

Master of Science

Jan 2021
Saint Louis, Missouri

Contents

List of Tables . . . . . **iv**

List of Figures . . . . . **v**

Acknowledgments . . . . . **vii**

Abstract . . . . . **viii**

1 Introduction **1**
1.1 Motivation 2
1.1.1 Cost to Design and Train a Network 2
1.1.2 Black-box Model and Weight Distribution 2
1.1.3 Neural Network Model and Markov Chain 3
1.2 Hypothesis 4
1.3 Important Concepts Overview 4
1.3.1 Probabilistic SVMs 4
1.3.2 Model Construction Technique 4
1.3.3 Ensemble Learning 5

2 MLM Construction and Experiments **6**
2.1 CNN Choice 6
2.2 MLM Structure 6
2.2.1 Design Flow of the Machine Learning Model 7
2.2.2 Probabilistic Support Vector Machine and GiniSVM 12
2.2.3 Parameter Sampling 14
2.2.4 Overall MLM Structure 15
2.3 MLM Predicted CNNs 17
2.3.1 Training the MLM 17
2.3.2 Experiments: Predicting ResNet Models 17
2.4 Experiments: Noisy Predicted ResNet Models 29
2.4.1 Noise at the Inverse Sigmoid Sampling 29
2.4.2 Noise at the Initial Layer 30
2.4.3 Experiment I: Fixed Noise 30
2.4.4 Experiment II: Gaussian Noise 33
2.5 Results 35

3 Ensemble Models **36**
3.1 Ensemble Methods 36
3.1.1 Plurality Voting 37
3.1.2 Averaging 37
3.2 Experiments and Results 38
3.2.1 Experimental Setups 39
3.2.2 Experiments I: Ensemble of ResNet32 and ResNet56 39
3.2.3 Experiment II: Rethink the Training Layers 42

4 Conclusion and Future works **44**
4.1 Conlusion 44
4.2 Future Works 45
4.3 Reference 46

List of Tables

2.1 Hyper-parameters values for the first grid search on ResNet32 and ResNet56 and pretrained networks accuracy 19
2.2 Best predicted network accuracy and their hyper-parameter settings 19
2.3 Experiment setting: Fixed noise to initial layer and sampling process 31
2.4 Experiment setting: Gaussian noisy with different σ to initial layer(NM) and sampling process(NS) 33

3.1 Performance(%) of baseline model implemented to generate the noisy model 38
3.2 Experiment settings for ResNet Noisy ensemble 39
3.3 Experiment settings for ResNet Noisy ensemble 42
3.4 Best performance with this experiment so far 43

4.1 Simple test results for parameter compression 45

List of Figures

1.1 Simple Markov state transition 3

2.1 ResNet basic building block: Two convolutional layers with a short-cut connection(Image from the original paper[7]) 7
2.2 A diagram of two connected convolutional layer 8
2.3 Two connected convolutional layers in ResNet 8
2.4 Visualization of data-flow when j = 0 10
2.5 Visualization of data-flow when j = 1 11
2.6 GiniSVM training results. Original 2D data(Top Left). Probability margin with $\gamma = 5$(Top right). Probability margin with $\gamma = 0.5$(bottom) 15
2.7 Graphs for structure of $G_(j = 0)$ for $K = M$(left) and $K = 2M$(right) 17
2.8 A block view of ResNet-32 and ResNet-56, blue parts are the convolutional layers for the MLM 18
2.9 Grid search results of gamma and Ks for predicted ResNet32 19
2.10 Grid search results of gamma and Ks for predicted ResNet56 20
2.11 Inference class confusion matrix of original ResNet32(left) and predicted ResNet32(right) from MLM with $\gamma = 0.08$ and ks = 800 20
2.12 Inference class confusion matrix of original ResNet56(left) and predicted ResNet56(right) from MLM with $\gamma = 0.08$ and ks = 800 21
2.13 Inference class confusion matrix of original ResNet32 and predicted ResNet32 from MLM with $\gamma = 0.08$,1,5 and ks = 800 22
2.14 Inference class confusion matrix of original ResNet56 and predicted ResNet56 from MLM with $\gamma = 0.5$,1,5 and ks = 800 23
2.15 Inference class confusion matrix of original ResNet56 and predicted ResNet56 from MLM with $\gamma = 0.08$ and ks = 800,10,1 23
2.16 MAE and RMSE of predicted ResNet32 for $\gamma = 0.08$ and $ks = 800$ 25
2.17 MAE and RMSE of predicted ResNet56 for $\gamma = 0.08$ and $ks = 800$ 25
2.18 Percentage scaling(%) of predicted ResNet32 for $\gamma = 0.08$ and $ks = 800$ . . . 26
2.19 Percentage scaling(%) of predicted ResNet56 for $\gamma = 0.08$ and $ks = 800$ . . . 26
2.20 Percentage scaling(%) of predicted ResNet32 for $\gamma = 1$ and $ks = 800$ 27
2.21 Percentage scaling(%) of predicted ResNet56 for $\gamma = 1$ and $ks = 800$ 27
2.22 s^{-1} for fixed noise(left) and Gaussian noise(right) 30
2.23 Performance of ResNet32 with fixed NS 31
2.24 Performance of ResNet56 with fixed NS 32
2.25 Performance of ResNet32 and ResNet56 with fixed NM 32

2.26 Performance of ResNet32 with Gaussian NS 33
2.27 Performance of ResNet56 with Gaussian NS 34
2.28 Performance of ResNet56 with Gaussian NM 34

3.1 Structure of ensemble of CNNs with plurality vote 37
3.2 Structure of ensemble of CNNs with Averaging 38
3.3 Accuracy for ensemble models with NS noise for experiment I 40
3.4 Accuracy for ensemble models with NM noise for the experiment I 41
3.5 Accuracy for ensemble models with NS noise with training the CL 64 and fully connected layer. 43
3.6 Accuracy for ensemble models with NS noise with training the CL 64 and fully connected layer. 43

Acknowledgments

I want to give my sincere gratitude to to Dr. Shantanu Chakrabartty for guiding and supporting me on this one-year thesis research. And I am grateful to all the supports from Oindrila Chatterjee.

A special thanks goes to the committee members for reviewing the thesis and providing valuable feedbacks to me.

Yuqiu Zhang

Washington University in Saint Louis
Jan 2021

ABSTRACT OF THE THESIS

Investigating Patterns in Convolution Neural Network Parameters Using Probabilistic Support Vector Machines

by

Yuqiu Zhang

Master of Science in Electrical Engineering

Washington University in St. Louis, Jan 2021

Research Advisor: Dr. Shantanu Chakrabartty

In the last decade, artificial neural networks(ANNs) and their deeper variants have been widely used in various disciplines and fields. Among different types of deep neural networks, the convolutional neural network(CNN) in particular has demonstrated excellent performance on a variety of image recognition tasks, and on a range of complex datasets. However, in spite of their widespread use, CNNs are still essentially black-box models. Though existing research shows that the weights in the kernels follow some distribution, we are unable to establish a formulaic relationship between the variation of parameters and the predicted results. Additionally, the contribution of the underlying parameters in different layers to the final model performance is still not fully understood. In my thesis, I explore if the distribution of parameters of any convolutional layer is related to the corresponding distribution in the previous layer, and if this distribution is learnable and has a relationship with the classification task at hand. With this hypothesis, I propose a Markov chain-based approach for modeling the structural patterns between subsequent layers of the CNN using probabilistic

support vector machines(PSVMs) due to limitations in the training data size for learning the weights. Given a baseline trained CNN model, the PSVM approach is capable of generating multiple CNN models using the inferred distributions, such that each model gives a distinct inference to the classification problem. Additionally, I explored the possibility of using an ensemble of the generated CNNs for solving the original classification task. Experiments using a ResNet-based architecture as the baseline on CIFAR-10 datasets showed encouraging results on both the individual model as well as the ensemble case.

Chapter 1

Introduction

Artificial neural networks(ANNs) are recognized as high-performance models for classification problems. They have proved to be efficient tools for many of today's applications like automatic driving, image and video recognition and restoration, big-data analysis. However, high performance deep neural networks have millions of parameters, and the iterative training procedure thus involves a very high computational cost. There are methods to prune and compress the models to reduce their computational burden[13,14,15]. Some recent works have proposed different methods that can automate the design and compression process. Neural architecture search is a method that uses Machine Learning(ML) technique to automatically find an optimal ANN architecture[16, 17, 18, 19]. It is proved to have the ability to generate ANNs that are superior to those designed traditionally by humans for certain problems. Though the automatic neural network design techniques are gradually maturing, since the model contains a huge amount of parameters, at the current stage, researchers are unable to conclude a precise and systematic analysis of the relations between different parameters in a network.

This research attempts to study the relationships between parameters in convolutional neural networks(CNNs). I assume there exists a certain relation between adjacent convolutional layers and proposed a machine learning model(MLM) that can be trained to represent this relation. The MLM's generalization ability is evaluated by the model it creates based only on the knowledge of the initial layer. Experiments and results show that the MLM is able to generate a CNN that has very similar performance but different in parameters. In addition, taking advantage of the difference, I insert noise when creating CNNs from the MLM and use ensemble methods to increase the performance on original classification problems.

1.1 Motivation

1.1.1 Cost to Design and Train a Network

Traditionally, a deep neural network is designed based on experience. The design process involves choosing different hyper-parameters like block size, channel dimension, wiring, parameter initialization, etc. Researches in recent years have successfully automated the design process of ANNs. For example, Neural Architecture Search(NAS)[16] proposed in 2016 automates the process by defining a search space, search strategy and evaluation technique, and iteratively optimizing the network. However, either human-power or automatic design requires retraining the network. It is hard to efficiently adjust the network by tuning the parameters without repeatedly going through forwarding and back-propagation steps. With deeper models and larger datasets, time and computational cost become a burden. Inspired by this drawback, I made a very simple assumption: What if there exists a certain pattern or relation between parameters in the network? During the training of a neural network, a relation between parameters is formed along with their update. So, if there exists a model that can express the relationship between parameters in different layers, it is possible to speed up the training process of a model. For example, adjust parameters in a single layer and then use the novel model to tune parameters in subsequent layers.

1.1.2 Black-box Model and Weight Distribution

The Block-box model is a type of model where inputs and outputs are only functionally connected. Researchers have the ability to define the model, input, output as well as the training procedure. However, the relationship between parameters and inference can not be physically and intuitively explained by words or equations. Many machine learning models, including CNN, are widely regarded as black-box models. In the area of machine learning, researchers believe that the explainability of the model is decreasing as the accuracy of the model is increasing. Deep neural networks are considered the least explainable model due to its high number of parameters and complex internal connections[1].

There is some recent research relative to the distribution of weights in CNNs. In this paper[10], the researchers proposed a method to prune channels based on the distribution of weights in kernels. They use statistical tests to show that the distribution of filter weights in VGG-16 and ResNet18 are approximately gaussian. In the paper[11], the researchers implement a way to adjust the amount of update based on the current distribution of weights.

1.1.3 Neural Network Model and Markov Chain

The Markov Chain is a stochastic model where for a sequence of states(or events), the probability of transiting to the next state depends only on the current state. A simple state transition is shown in Figure 1.1.

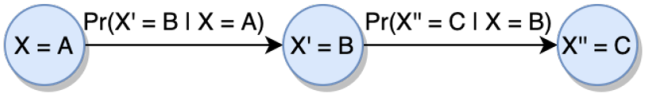

Figure 1.1: Simple Markov state transition

The structure of a CNN can be described by a graph model. Suppose a CNN is a graph model G and every layer is a node. The forward data-flow is represented by directed edge. Then the graph model G is a direct acyclic graph where the out-going edges of a node is dependent on all its in-going edges and the node itself. Based on the dependency of the data-flow, I assume there exists a relation between two nodes of an edge, and transfer this relation into an Markov model: For these two nodes, the probability of transiting to the node that the edge pointing to is based on the node that the edge starting from. Since a node is a convolutional layer, I replace the node with filter weights in the convolutional layer: For two adjacent convolutional layer, there exist $P(W_i \mid W_{i-1})$, W_i is the weight in the deeper layer and W_{i-1} is its previous layer. If a model can generate $P(W_i \mid W_{i-1})$ for every two connected layers, then it is possible to rebuild or tune the model without going through training sets.

Since, the existence of patterns between filter weights in two adjacent convolutional layer is not theoretically proved, in this research, I have the following hypothesis.

1.2 Hypothesis

In convolutional neural networks, there exist a relation or pattern between filter weights for every two connected convolutional layers, and the relation or pattern satisfies: (i)It is learnable. (ii) It has connection to the problem. The first condition means that the relation or pattern is not completely random. The second condition means that the relation is dependent on the problem.

1.3 Important Concepts Overview

1.3.1 Probabilistic SVMs

A Support Vector Machine(SVM) is a classical machine learning algorithm for solving classification problems[2]. Traditional SVMs find a margin between different labels. A Probabilistic Support Vector Machine(PSVM), on the other hand, gives the conditional probability of output labels based on input data[3, 4]. PSVM is the basic component of the MLM proposed in this thesis. The advantage of using PSVM in the MLM model is its relatively good generalization ability over small datasets.

1.3.2 Model Construction Technique

The MLM is constructed based on forwarding data-flow of CNNs. To simplify the problem, the MLM is designed to only learn the distribution of parameters in convolutional layers. The parameters in batch normalization layers and fully connected layers remain unchanged in the output model from MLM. The MLM structure is similar to CNNs where it consists of many layers of PSVMs. Each layer of PSVMs is trained by two connected convolutional layers in the baseline CNN. Therefore, the well-trained MLM is able to predict the whole CNN model only from the parameters in the first convolutional layer.

1.3.3 Ensemble Learning

Ensemble learning is a machine learning technique that uses several distinct machine learning models, or "weak learners", to create a stronger model to solve a problem[5]. This technique is applicable to CNN models. For example, researchers combined a few CNNs with the K-nearest neighbor models to increase the accuracy of CIFAR-10 prediction[6]. For my thesis, I implemented the CNNs generated by MLM as "weak learners" to create ensemble models for image classification problems. The "weak learners" are generated by adding noise to either sampling procedure or initial layers to create CNNs that are slightly different to each other. The full setting and experiment process are illustrated in the Ensemble Model section.

Chapter 2

MLM Construction and Experiments

This chapter will illustrate the procedures to construct the Machine Learning Model(MLM) for a particular type of CNNs, experimental setups, and results.

2.1 CNN Choice

I establish my basic experiments on the ResNet that was designed for the CIFAR-10 classification [7,9]. The reason I chose ResNet is its representativeness and conciseness. The network is widely used as a benchmark for many of today's state-of-the-art image recognition CNNs. The structure and training process, as suggested in its original paper, is relatively simple. It does not include complex wiring among layers, and it has no extra modification on pooling layers or batch normalization. Therefore, using the ResNet can minimize possible disturbances to the experiments that are based only on convolutional layers.

2.2 MLM Structure

MLM contains a set of functions that each learns the relation of every two adjacent convolutional layers. Denote a single function as $L(X)$. For a single $L(X)$, there exists $W_i = L(W_{i-1})$. i is the index of a convolutional layer, and W is a set of all filter weights in that layer. From my hypothesis, the probability of having a set of weights in convolutional layers follows the Markov Chain(i.e., the probability of a layer i has a weight W_i depends

and only depends on the weights in layer $i-1$). This can be written as a conditional probability $P\left(W_i \mid W_{i-1}\right)$. The function $L(X)$ will learn this relation from a pretrained CNN with $P\left(W_i \mid W_{i-1}\right)$ and W_{i-1}. The sigmoid function will sample the probability and weight value. The method to learn the relation is the probabilistic support vector machine(PSVM). The structure of the MLM is designed based on the forwarding data-path of the ResNet[7]. The detailed design flow and reasons will be illustrated in the next few sections.

2.2.1 Design Flow of the Machine Learning Model

To design the structure of the MLM, I started from the forwarding data-path of the ResNet. The ResNet is composed of certain numbers of convolutional blocks (Conv blocks) plus an input convolutional layer and a fully connected layer. There is an unweighted shortcut connection from the input of a block to the output, as shown in Figure 2.1.

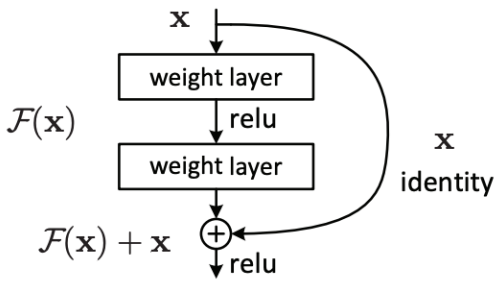

Figure 2.1: ResNet basic building block: Two convolutional layers with a short-cut connection(Image from the original paper[7])

Since the assumption is "weight relation follows Markov Chain", every $L(X)$ is independent to each other. Suppose for any two connected convolutional layers, there are two sets of weights $W_1 : \mathbb{R}^n$ in the convolutional layer(CL)1, $W_2 : \mathbb{R}^m$ in the CL2. Define a set of probability P where for every $\hat{w} \in W_2$, there exist a probability

$$p \in P : p\left(\widehat{w} \mid \mathrm{x} = \{w_i : w_i \in W_1; i = 0, 1, \ldots, q; q \leq n-1\}\right) \tag{2.1}$$

Notation. In all sections, denote w as a weight parameter in the CL1 without any superscript. Denote any true weight parameter in any CL2 by $\hat{w}$. Denote any predicted weight parameter in any CL2 by $\bar{w}$.

To convert between probability value p and weight value w, define a function S where $S(w) = p$ and $S^{-1}(p) = w$. Here $w \in \mathbb{R}$ is a scalar weight value and $p : 0 \leq p \leq 1$ is a scalar probability value. The next step is to find a proper x that may have relation with $\hat{w}$. I start from the forward data-flow of two convolutional layers.

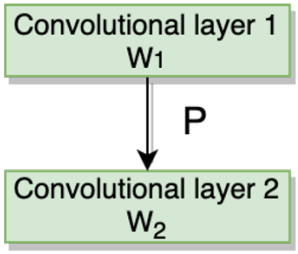

Figure 2.2: A diagram of two connected convolutional layer

As shown in Figure 2.3, define the output of the first convolutional layer as $F_1(X)$ and the output of the second convolutional layer as $F_2(X)$. $F_1(X)$ and $F_2(X)$ with input data X are shown in Equation 2.2. A_1 and A_2 are activation functions, BN_1 and BN_2 are batch normalizations. Q_1 and Q_2 are 2D cross-correlation functions. b_1 and b_2 are biases.

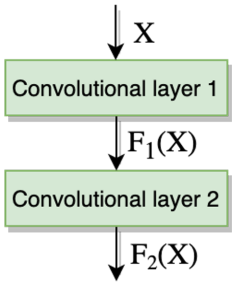

Figure 2.3: Two connected convolutional layers in ResNet

$$\begin{aligned} F_1(X) &= A_1\left(BN_1\left((Q_1(X) + b_1)\right)\right) \\ F_2(X) &= A_2\left(BN_2\left((Q_2\left(F_1(X)\right) + b_2)\right)\right) \end{aligned} \tag{2.2}$$

Since batch normalizations, activations, and bias terms all do not influence the data-path, to simplify the problem, I ignore them and only focus on Q_1 and Q_2, which are the calculations involving filter weights. Suppose the CL1 has M input channels and N output channels. The CL2 has N input channels and K output channels. Q_1 and Q_2 can be expressed as Equation 2.3. Here, f_j^1 and f_h^2 are outputs of a single output channel j and h.

$$\begin{aligned} Q_1(X) &= \left\{ f_j^1(X), j = 0, 1, \ldots, N-1 \right\} \\ Q_2\left(X'\right) &= \left\{ f_h^2\left(X'\right), h = 0, 1, \ldots, K-1; X' = F_1(X) \right\} \end{aligned} \tag{2.3}$$

The equation for a single output channel can be expressed as Equation 2.4. Here, w_{ij} and $\hat{w}_{jh}$ are single 3×3 filters in the CL1 and CL2 (All filters in the ResNet are 3×3, which contain 9 parameters). x_i is the input data of size $H \times W$ and x'_j is the input data of size $H' \times W'$. $\star$ is a 2D cross-correlation operation.

$$\begin{gathered} f_j^1(X) = b_j + \textstyle\sum_{i=0}^{M-1} w_{ij} \star x_i \\ f_h^2\left(X'\right) = b_h + \textstyle\sum_{j=0}^{N-1} \widehat{w}_{jh} \star x'_j \\ X = \left\{ x_i : \mathbb{R}^{H \times W}, i = 0, 1, \ldots, M-1 \right\}; X' = \left\{ x'_j : \mathbb{R}^{H' \times W'}, j = 0, 1, \ldots, N-1 \right\} \end{gathered} \tag{2.4}$$

Exploring the relation. The key relation is that x'_j is the result of $f_j^1(X) + b_j$ going through batch normalization and activation. Combining with how $f_j(x)$ is calculated by w_{ij}, it is obvious to conclude that for every $j = J, J = 0, 1, ..., N-1$, $w_{i(j=J)}$ and $\hat{w}_{(j=J)h}$ are on the same datapath J. The visualization of this relation is shown in Figure 2.4 and Figure 2.5. The green 3 by 3 filters are w_{ij} and $\hat{w}_{jh}$ when $j = 0$(Figure 2.4) and $j = 1$ (Figure 2.5).

Then, the relation between these two convolutional layers can be expressed as a set of probability $P = \left\{ \mathrm{p}_j \right\}, j = 0, 1, 2, \ldots, N-1$. And for j equals to any fixed value J, $\mathrm{p}_{j=J}$ is shown in Equation 2.5. Here $h = 0, 1, 2, ..., K-1$, $i = 0, 1, 2, ..., M-1$ where K is the number of output channels in the CL2 and M is the number of input channel in the CL1 and $t = 0, 1, 2, ..., 8$ is the index of a flattened 3 by 3 filter. $p_{(j=J)ht}$ is a scalar probability value.

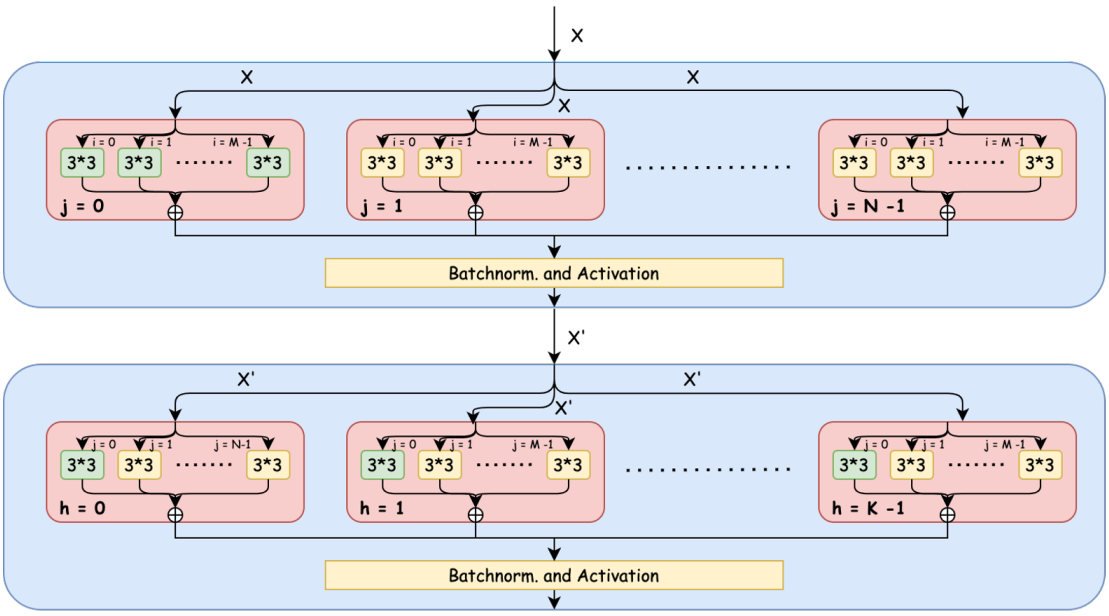

Figure 2.4: Visualization of data-flow when j = 0

$$\mathrm{p}_{j=J}\left(\widehat{w}_{(j=J)h} \mid w_{i(j=J)}\right) = \left\{p_{(j=J)ht}\left(\widehat{w}_{(j=J)ht} \in \widehat{w}_{(j=J)h} \mid w_{i(j=J)}\right)\right\} \tag{2.5}$$

Notice that all p_j are independent to each other due their unique data-paths. So for each $j = J$, I select a proper machine learning method to generalize the relation individually. Denote the set of machine learning method for all $j = J$ by L, which has $L = \{G_j\}, j = 0, 1, ..., N-1$. G_j is the set of functions for an individual relation. To simplify the problem, instead of leaning the relation for a 9-dimensional data with a single function, I divided the function G_j into 9 different sub-functions as $G_j = \{g_{jt}\}, t = 0, 1, ..., 8$ to learn parameters in the 3×3 filter separately.

Here, g_{jt} is a single machine learning function that will be trained to predict a parameter $\hat{w}_{jht}$ from a set of parameters w_{ij}, as shown in the Equation 2.6. The function g includes two part as shown in the Equation 2.7: (i) A function f that predicts the conditional probability of a parameter $p(\hat{w})$ from a set of parameters w. (ii) A function s that samples the true weight from the probability $p(\hat{w})$.

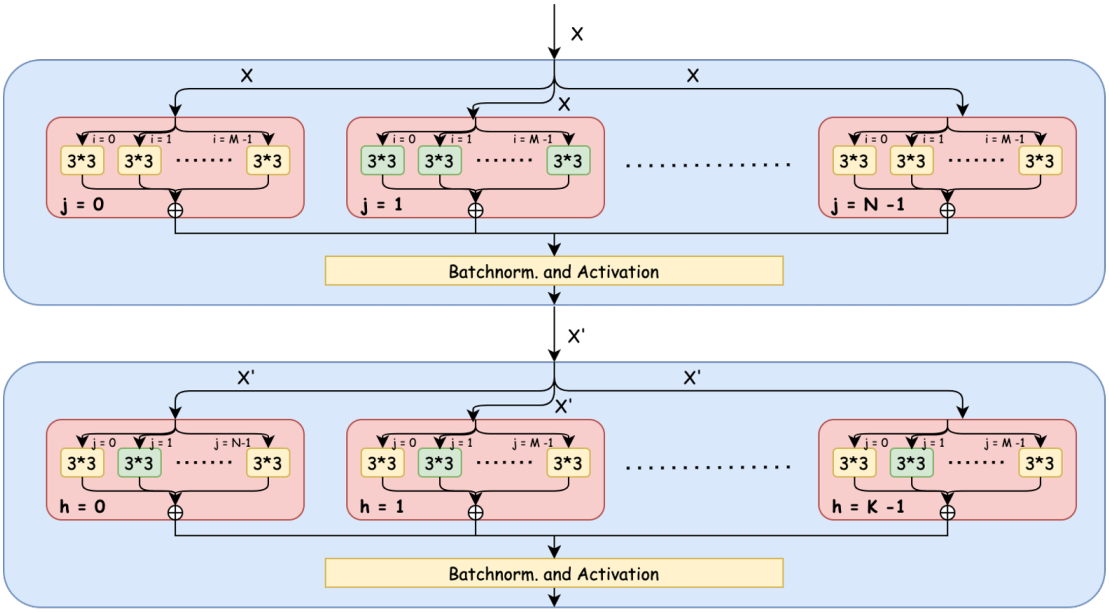

Figure 2.5: Visualization of data-flow when j = 1

$$g_{(j=J)t}\left(w_{i(j=J)}\right) = \bar{w}_{(j=J)ht} \tag{2.6}$$

$$g_{(j=J)t}\left(w_{i(j=J)}\right) = s(f_t(w_{i(j=J)})) = \bar{w}_{(j=J)ht} \tag{2.7}$$

Machine learning method. The function f is chosen based on the characteristic of the dataset and labels: First, $w_{i(j=J)}$ is a dataset of a size $M \times 9$ and $p_{j=J}(\widehat{w}_{(j=J)h})$ has a size $K \times 1$. Both M and K are channel numbers(dimension) of a convolutional layer. For ResNet32 and ResNet56, the size is from 16 to 64, and for wider CNNs, the size could raise to 512 to 2048. This suggests that the size of a training set can be limited. Therefore I exclude the machine learning methods that require a large volume of data to fit. Besides, the training set has a dimension of 9, and the label is a probability value. Based on all the above points, I adopt the GiniSVM, as a type of Probabilistic Support Vector Machine(PSVM), as the function f. GiniSVM has a great generalization over small and multidimensional datasets.

2.2.2 Probabilistic Support Vector Machine and GiniSVM

In this research, the probability of weights of the current convolutional layer based on the weights of its previous convolutional layer is learnt by the Gini Support Vector Machine (GiniSVM). The reason to choose the PSVM is described in the previous section. Usually, a PSVM gives the probabilities of the input to its corresponding output labels. It can be expressed as:

$$\begin{aligned} &\bar{Y} = f(X) \\ &\bar{Y} = \{\bar{y}_0, \bar{y}_1, \cdots, \bar{y}_i\}, i \geq 1 \\ &X = \{x_0, x_1, \cdots, x_j\}, j \in Z^+ \end{aligned} \tag{2.8}$$

X is the input data, $\bar{Y}$ is the output probabilities, f is the trained PSVM. The output should contain at least two classes, so $i \geq 1$. X, the input dataset, in theory, has no limit on the number of dimensions. To implement on kernel weights, the real number parameters Y are sampled to probabilities during training, and predicted probabilities $\hat{Y}$ is sampled back to real number parameters during inference.

GiniSVM.[1] GiniSVM is a probabilistic support vector machine that based on Gini entropy. Suppose there is a training set $\{x_i\}$ with $i = 1, ..., N$ and prior probabilities $Y = \{y_{ik} = P(C_k|x_i)\}$. C_k is class labels where $k = 1, 2, ..., M$. The method of the Gini PSVM is to minimize the Distance matrixes between prior knowledge Y and output probability $P = P_k(x)$, as Equation 2.9[4]. $D_Q : \mathbb{R}^M \times \mathbb{R}^M$ is the distance matrix of output and piror knowledge, and $D_I : \mathbb{R}^M \times \mathbb{R}^M$, as a regularization term, evaluate the distance between the output and a uniform distribution U. U is denoted by $U(x_i) = 1/M$ where M represents the total number of labels. γ controls the amount of regulation from the matrix D_I.

$$\min_P G(P) = \min_P [D_Q(Y, P) + \gamma D_I(P, U)] \tag{2.9}$$

The minimization subjects to following constrains:

[1]This subsection is based on[4], the equations in rest of the section refer from the original paper

$$\begin{gathered}\sum_{i=1}^{N} P_k\left(\mathbf{x}_i\right)=\sum_{i=1}^{N} y_{ik}, \quad k=1, \ldots M \\ P_k(\mathbf{x}) \geq 0 \\ \sum_{k=1}^{M} P_k\left(\mathbf{x}_i\right)=1\end{gathered} \tag{2.10}$$

The distance matrix D_Q is calculated by:

$$D_Q(\hat{P}, P)=\frac{C}{2} \sum_{k=1}^{M} \sum_{\mathbf{x}, \mathbf{v} \in \mathcal{T}} K(\mathbf{x}, \mathbf{v})\left[\hat{P}_k(\mathbf{x})-P_k(\mathbf{x})\right]\left[\hat{P}_k(\mathbf{v})-P_k(\mathbf{v})\right] \tag{2.11}$$

Here, $K(\mathbf{x}, \mathbf{v})$ is the kernel function, and in the following experiments, all the kernels are RBF kernel as Equation 2.12. ks is the kernel scale, which is an important hyper-parameter in later experiments.

$$K(\mathbf{x}, \mathbf{v})=exp(-ks||\mathbf{x}-\mathbf{v}||^2) \tag{2.12}$$

The distance matrix D_I is calculated based on Gini entropy:

$$\begin{aligned} D_I(P, U) &=\frac{1}{2} \sum_{k=1}^{M} \sum_{\mathbf{x} \in \mathcal{T}}\left(P_k(\mathbf{x})-U_{ik}\right)^2 \\ &=\frac{1}{2} \sum_{k=1}^{M} \sum_{\mathbf{x} \in \mathcal{T}} P_k(\mathbf{x})^2+\mathrm{cst} \end{aligned} \tag{2.13}$$

The final cost function with dual formulation:

$$H_g=\sum_{k=1}^{M}\left[\frac{1}{2C} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_k^i Q_{ij} \lambda_k^j+\frac{\gamma}{2} \sum_{i=1}^{N}\left(y_{ik}-\lambda_k^i / C\right)^2\right] \tag{2.14}$$

$$Q_{ij}=K\left(\mathbf{x}_i, \mathbf{x}_j\right) \tag{2.15}$$

The cost function subjects to constrains:

$$\begin{aligned} \sum_{k=1}^{M} \lambda_k^i &= 0, \quad i = 1, \dots N \\ \sum_{i=1}^{N} \lambda_k^i &= 0, \quad k = 1, \dots M \\ \lambda_k^i &\leq C y_{ik} \end{aligned} \tag{2.16}$$

Hyper-parameters. In this research, the regularization parameter γ and kernel scale ks are two important hyper-parameters that will be adjusted during experiments. γ controls the regularization during training. A smaller γ means the GiniSVM is more likely to be over-fitted to the training set. An example with different γ value is shown in Figure 2.6. This is an equal probability contour plot for the top left dataset. White areas are the probability margin, and the dashed lines are 50% probability level. As γ decreases, the GiniSVM is more fitted to the original dataset(White area becomes smaller and sharper around cycle data-points). ks is the hyper-parameter that based on the scale of input data. Its influence will be illustrated in the Experiment section with a grid-search method.

2.2.3 Parameter Sampling

The parameters are sampled to probabilities by the sigmoid function(logistic function), as in Equation 2.17. β is a scaling scalar and ns is a noise parameter that will be used later in the experiment. Normally, β equals is set to 1 and ns is set to zero. A single GiniSVM is designed to predict a weight parameter in a 3×3 kernel. Then Y, the probability vector, for this GiniSVM contains two probabilities value $\{p, 1-p\}$. This means that the GiniSVM learns the probability that an input data belongs to one of the two labels, and the probability of a label to be the first label is sampled by Equation 2.17 that based on a weight value x. During the inference stage, the output probability value p will be sampled back to a real number by inverse sigmoid function(Equation 2.18). ns is the noise parameter that represents the mean of output weight variables x. In later experiments, I will slightly modify ns to generate different predictions for a single MLM.

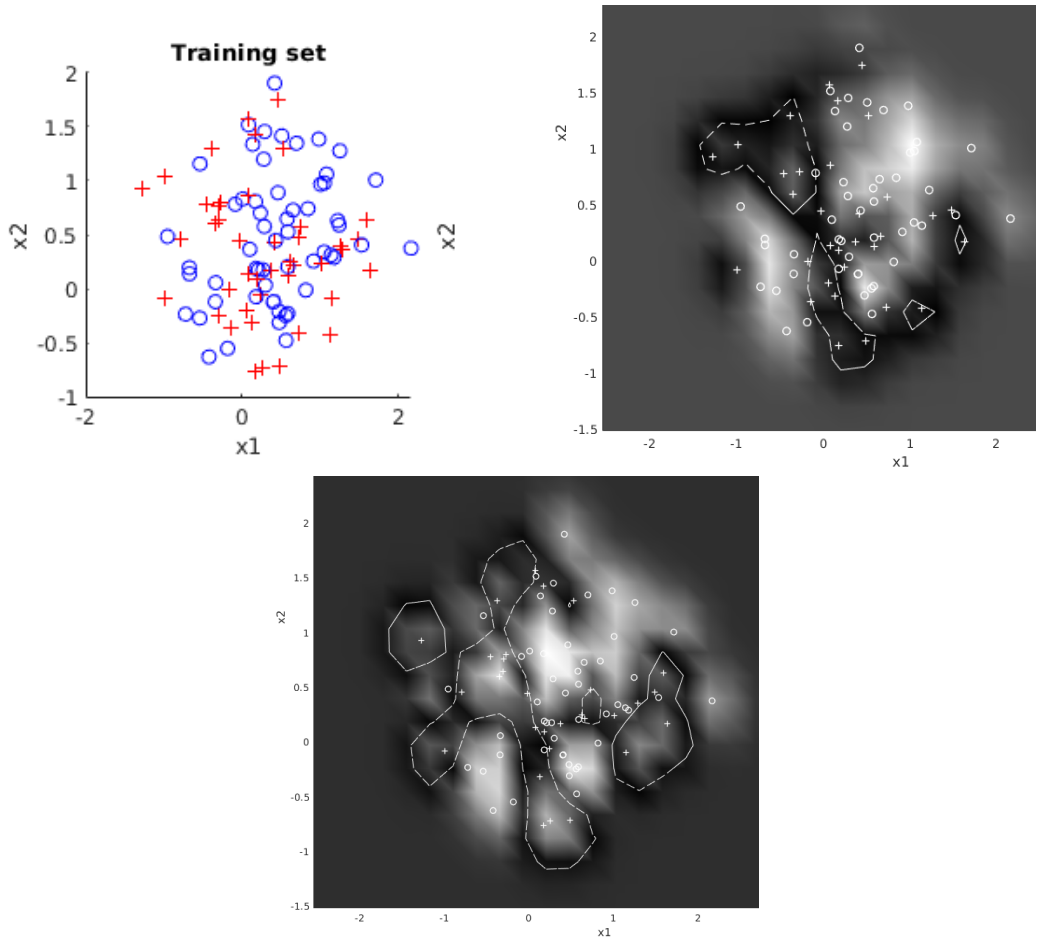

Figure 2.6: GiniSVM training results. Original 2D data(Top Left). Probability margin with $\gamma = 5$(Top right). Probability margin with $\gamma = 0.5$(bottom)

$$p = s(x) = \frac{1}{1 + e^{-\beta(x-ns)}} \tag{2.17}$$

$$\hat{y} = s^{-1}(x) = \frac{1}{\beta}(log(p) - log(1-p)) + ns \tag{2.18}$$

2.2.4 Overall MLM Structure

The MLM contains a set of function L for every pair of connected convolutional layers. As described in the previous three sections, for a single pair, $L = \{G_j\}, G_j = \{g_{jt}\}, j = 0, 1, 2, ..., N-1$. For every $j = J$ the function g_{jt} is described in Equation 2.7. In the equation,

$i = 0, 1, ..., M-1$ is the index of the input channel of the first layer, and $h = 0, 1, ..., K-1$ is the index of the output channel of the second layer. So the size of the input vector and label for a single g_{jt} are $M \times 9$ and $K \times 9$. In ResNet32 and ResNet56, there are two situations: (i) $K = 2M$ when two layers are raising in dimension. (ii) $K = M$ for reset of the layers. For (ii), the inputs and outputs already have a one to one correspondency. So suppose $K = M = A,\ a = 0, 1, ..., A-1$, then for a single $j = J$, $G_{j=J} = \{g_{(j=J)t}\}$.Here $t = 0, 1, ..., 8$ is the index of parameters in a flattened 3 by 3 filter. A single $g_{(j=J)t}$ is shown in Equation 2.19. A example of $L = \{G_j\}, j = 0$ is shown in Figure 2.7 left. The training set $\{(\text{Input}, \text{Probability})\}$for this situation is Equation 2.20 where $\widehat{w}$ is the true weight and s is the sampling equation in Equation 2.17 and Equation 2.18.

$$g_{(j=J)t}\left(w_{a(j=J)}\right) = s(f_t(w_{a(j=J)})) = \bar{w}_{(j=J)at} \tag{2.19}$$

$$\left\{\left(w_{a(j=J)}, \left\{s^{-1}\left(\widehat{w}_{(j=J)at}\right), 1 - s^{-1}\left(\widehat{w}_{(j=J)at}\right)\right\}\right)\right\} \tag{2.20}$$

For (ii), there are twice output vectors $\bar{w}_{(j=J)at}$ as input vectors $w_{a(j=J)}$. For this situation, I simply create two set of $G_{j=J}$ that each trained to predict a half of the outputs based on the same input vector. Denote $K = 2M, a = 0, 1, ..., M-1$. For s single $j = J$, $G^1_{j=J} = \{g^1_{(j=J)t}\}$ and $G^2_{j=J} = \{g^2_{(j=J)t}\}$, as shown in Equation 2.21. Then $L = \{G^1_j, G^2_j\}, j = 0, 1, ..., N-1$, see Figure 2.7 right. And the training set is Equation 2.22.

$$\begin{aligned} g^1_{(j=J)t}\left(w_{a(j=J)}\right) &= s(f_t(w_{a(j=J)})) = \bar{w}_{(j=J)at} \\ g^2_{(j=J)t}\left(w_{a(j=J)}\right) &= s(f_t(w_{a(j=J)})) = \bar{w}_{(j=J)(a+M)t} \end{aligned} \tag{2.21}$$

$$\begin{gathered} \left\{\left(w_{a(j=J)}, \left\{s^{-1}\left(\widehat{w}_{(j=J)at}\right), 1 - s^{-1}\left(\widehat{w}_{(j=J)at}\right)\right\}\right)\right\} \\ \left\{\left(w_{a(j=J)}, \left\{s^{-1}\left(\widehat{w}_{(j=J)(a+M)t}\right), 1 - s^{-1}\left(\widehat{w}_{(j=J)(a+M)t}\right)\right\}\right)\right\} \end{gathered} \tag{2.22}$$

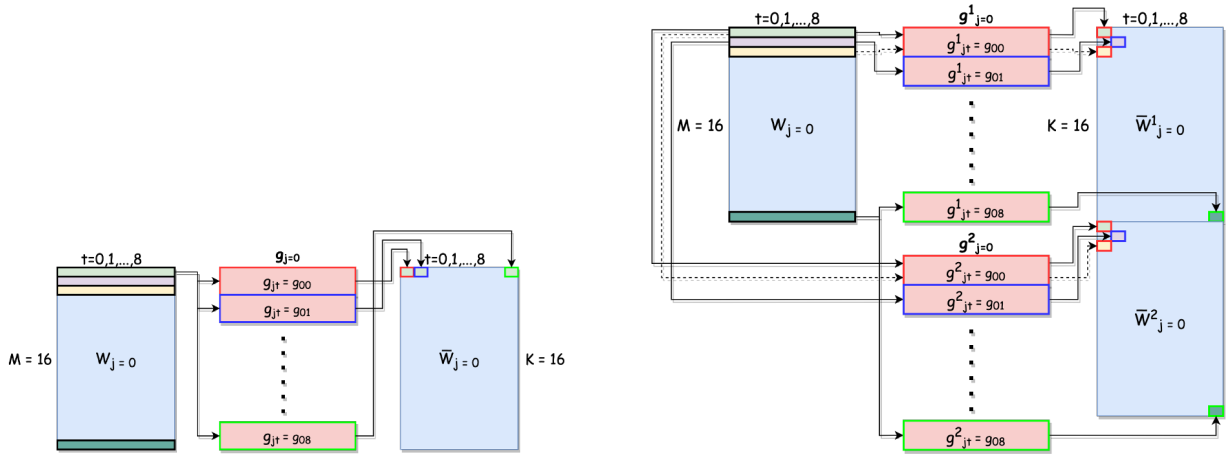

Figure 2.7: Graphs for structure of $G_(j = 0)$ for $K = M$(left) and $K = 2M$(right)

2.3 MLM Predicted CNNs

This section introduces how a MLM is trained to represent the relation of weights in the CNN, as well as the experiments to test the performance of the MLM.

2.3.1 Training the MLM

The MLM is first trained with the pretrained weights of ResNet32 and ResNet56. The above networks were trained on CIFAR-10 datasets[9] that match the accuracy in the paper[7]. The MLM training data are filters weights in specific convolutional layers, as blue boxes in Figure 2.8. The reason to exclude the initial 3 in 16 out convolutional layer is that this layer has much fewer dimensions than the subsequent one, which causes a high mismatch in the number of parameters between these two layers. To ensure the stability of the experiment, the model starts from the second convolutional layer.

2.3.2 Experiments: Predicting ResNet Models

Due to the influence of hyper-parameters γ and ks, in the first experiment, I implement a grid search with γ from 0.08 to 10 and ks from 1 to 800, as shown in Table 2.1. Each

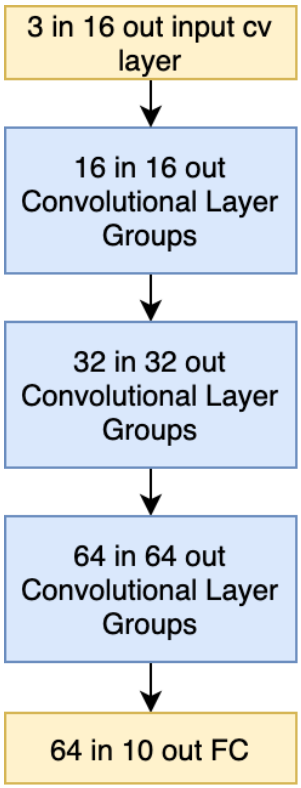

Figure 2.8: A block view of ResNet-32 and ResNet-56, blue parts are the convolutional layers for the MLM

MLM is trained by the procedures described in the previous sections. The trained MLMs are then tested based on the generated ResNets from an identical initial layer. The detailed procedures are: First, use the parameters of the first convolutional layer W_1 in the pretrained ResNet as an input to predict the parameters of the second convolutional layer $\bar{W}_2$. Then $\bar{W}_2$ becomes the input data to predict the parameters in the third layer $\bar{W}_3$. Repeat the procedure until the MLM predicts all parameters in the model. The predicted parameters are then imported to the original ResNet, and the new ResNet is named the "predicted ResNet." The predicted ResNet is then fine-tuned by freezing all layers except the final fully connected layer and training with the CIFAR-10 dataset for five epochs. The highest accuracy is recorded as the performance of the predicted ResNet. The model accuracy was shown in Figure 2.9 and Figure 2.10.

Results. The plots and table suggest that when $\gamma = 0.08$ and $ks = 800$, the predicted model can have a very similar performance to the pretrained model. The empirical results in

ks	1	2	5	10	50	100	200	400	600	800
γ	0.08	0.25	0.5	0.75	2	5	10			
Pretrained	ResNet32	92.63%								
Pretrained	ResNet56	93.52%								

Table 2.1: Hyper-parameters values for the first grid search on ResNet32 and ResNet56 and pretrained networks accuracy

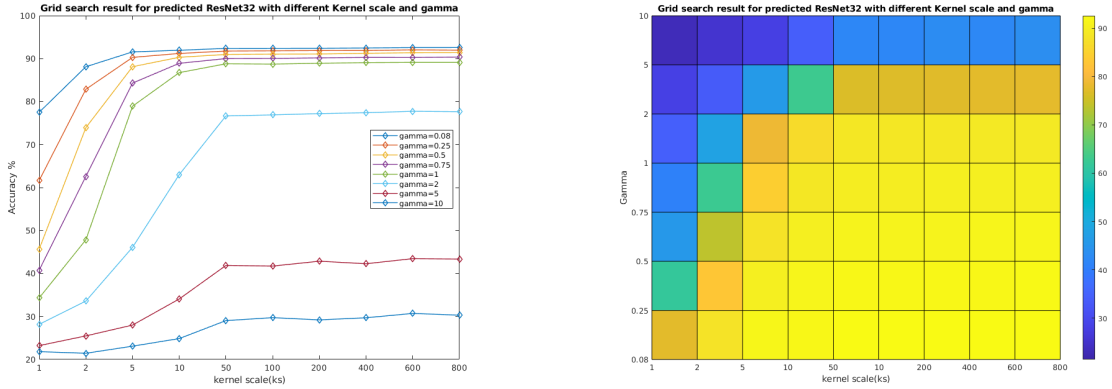

Figure 2.9: Grid search results of gamma and Ks for predicted ResNet32

Model	γ	ks	Predicted Acc	Pretrained Acc
ResNet32	0.08	800	92.58%	92.63%
ResNet56	0.08	800	93.43%	93.52%

Table 2.2: Best predicted network accuracy and their hyper-parameter settings

Figure 2.9 and Figure 2.10 show that the accuracy of the predicted ResNet is proportional to ks and inverse proportional to γ. This meets my exception since, first, kernel scale of the GiniSVM is $\frac{1}{2\sigma^2}$, as shown in Equation 2.23. Since the filters parameters are generally close to each other, for the RBF kernel, the variance σ should be set to a relative small value to properly calculate the similarity. Second, γ is the regularization term for the GiniSVMs[4]. It controls the trade-offs between distance matrix D_Q and D_I. High γ value will cause the GiniSVM to fit the probabilities more to the agnostic distance metric D_I, which can reduce overfitting but also negatively impacts the accuracy when the input vectors are close to weights in the pertained ResNet .

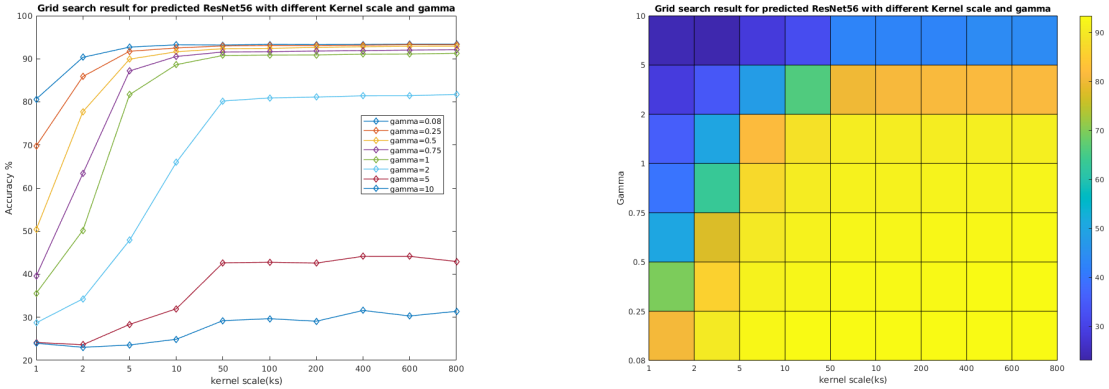

Figure 2.10: Grid search results of gamma and Ks for predicted ResNet56

$$K\left(x, x'\right) = \exp\left(-ks\left(x - x'\right)^{\top}\left(x - x'\right)\right) \tag{2.23}$$

Analysis. To better study the generalization ability of the MLM, I dig into the distribution of misclassifications of the validation dataset. Figure 2.11 and Figure 2.12 are the plots of misclassification confusion matrices for the pretrained ResNet and the predicted ResNet on Cifar10 validation dataset. This result shows that both models have a very similar distribution of errors when γ is low and ks is high. That means that for certain γ and ks, the MLM is able to reconstruct the original ResNet model that makes similar predictions with only the knowledge of the initial layer.

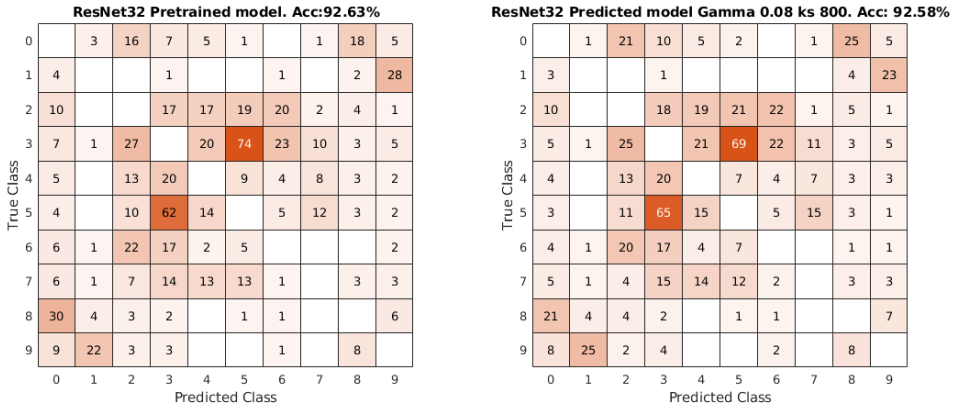

Figure 2.11: Inference class confusion matrix of original ResNet32(left) and predicted ResNet32(right) from MLM with $\gamma = 0.08$ and ks $= 800$

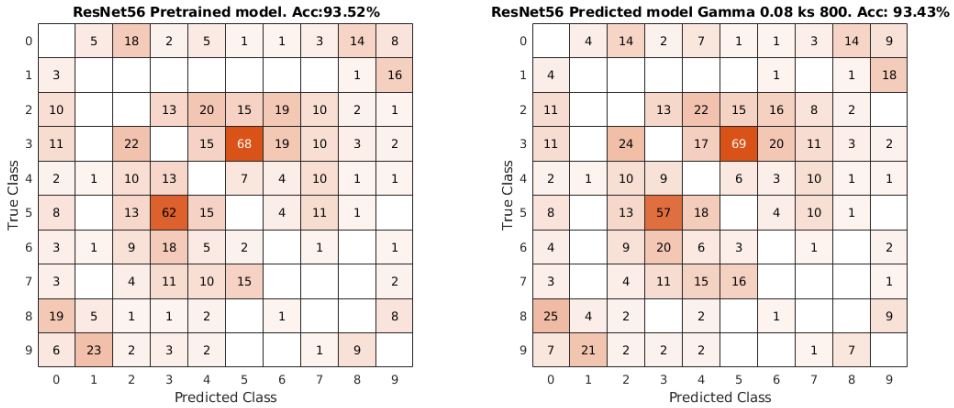

Figure 2.12: Inference class confusion matrix of original ResNet56(left) and predicted ResNet56(right) from MLM with $\gamma = 0.08$ and ks $= 800$

In addition to the predicted ResNet with the best performance, I also study the confusion matrices for ResNets generated by other γ and ks values. From Figure 2.13 to Figure 2.15, we can see that when γ is equals to 5 and ks is equals to 1, the performance of the model greatly deviates to the pretrained model. Although γ and ks are parameters that can prevent overfitting, their values should be limited to a range where the predicted model can still maintain sufficient attributes of the pretrained model.

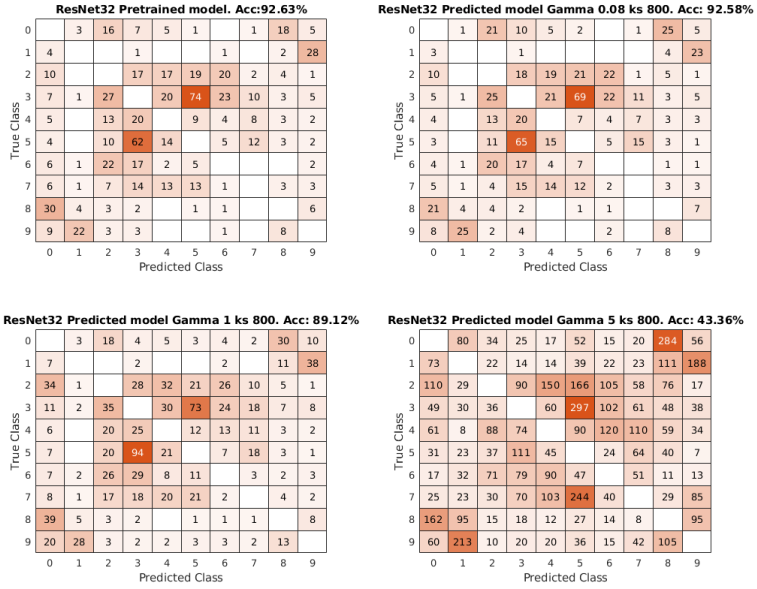

Figure 2.13: Inference class confusion matrix of original ResNet32 and predicted ResNet32 from MLM with γ = 0.08,1,5 and ks = 800

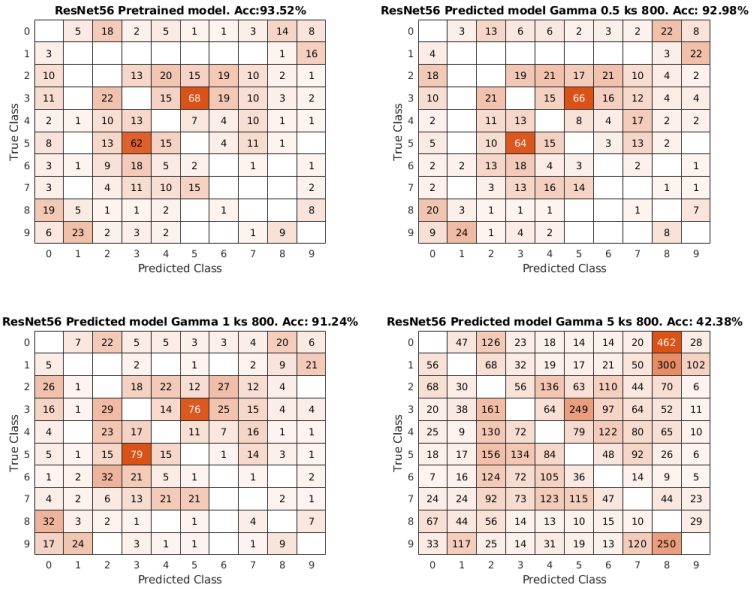

Figure 2.14: Inference class confusion matrix of original ResNet56 and predicted ResNet56 from MLM with $\gamma = 0.5,1,5$ and ks = 800

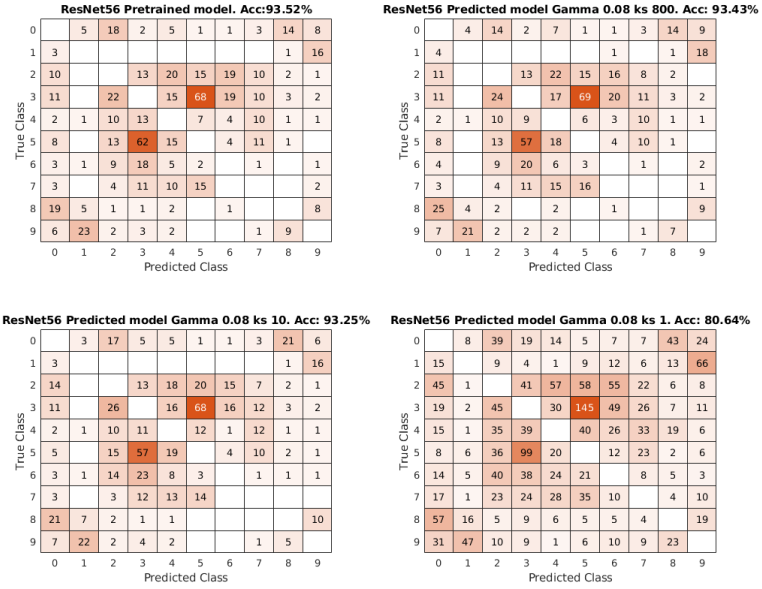

Figure 2.15: Inference class confusion matrix of original ResNet56 and predicted ResNet56 from MLM with $\gamma = 0.08$ and ks = 800,10,1

The validation results are not sufficient to study the MLM predicted models. In the thesis, I also studied the absolute distance and squared distance between predicted models and pretrained models. Since the number of parameters in a layer is large, I divide the parameters into different subsets by the output channel index and calculate the mean absolute error(MAE) and root mean squared error(RMSE) over these subsets(Equation 2.24 and Equation 2.25). In the two equations, j is the index of the output channels, i is the input channel's index, or can be interpreted as the index of a kernel(filter) in the output channel j. $\hat{w}_{ij}$ is a set of weights of a single 3×3 kernel(filter) of the pretrained ResNet. $\hat{w}_{ijt}$ is a single weight value where t is the parameter index in a flattened 3×3 filter. $\bar{w}_{ijt}$ is the corresponding weight in the predicted ResNet.

$$MAE_{ik} = \frac{1}{J} \sum_{j=0}^{J} |\hat{w}_{ijt} - \bar{w}_{ijt}| \tag{2.24}$$

$$RMSE_{ik} = \sqrt{\frac{1}{J} \sum_{j=0}^{J} (\hat{w}_{ijt} - \bar{w}_{ijt})^2} \tag{2.25}$$

I plot the distribution of MAE_{ik} and $RMSE_{ik}$ for every $i = 0, 1, ..., M-1$ and $t = 0, 1, ..., 8$. The box plots are shown from Figure 2.16 to Figure 2.17. In the plots, the MAE and RMSE reduce with increasing layer index, or depth of the network. For ResNet32 and ResNet56, the model dimension is doubled for every 10 and 18 layers. The result meets the expectation since the deeper network contains more training data for every GiniSVM. So the error, in theory, will reduce with the increase of training set size.

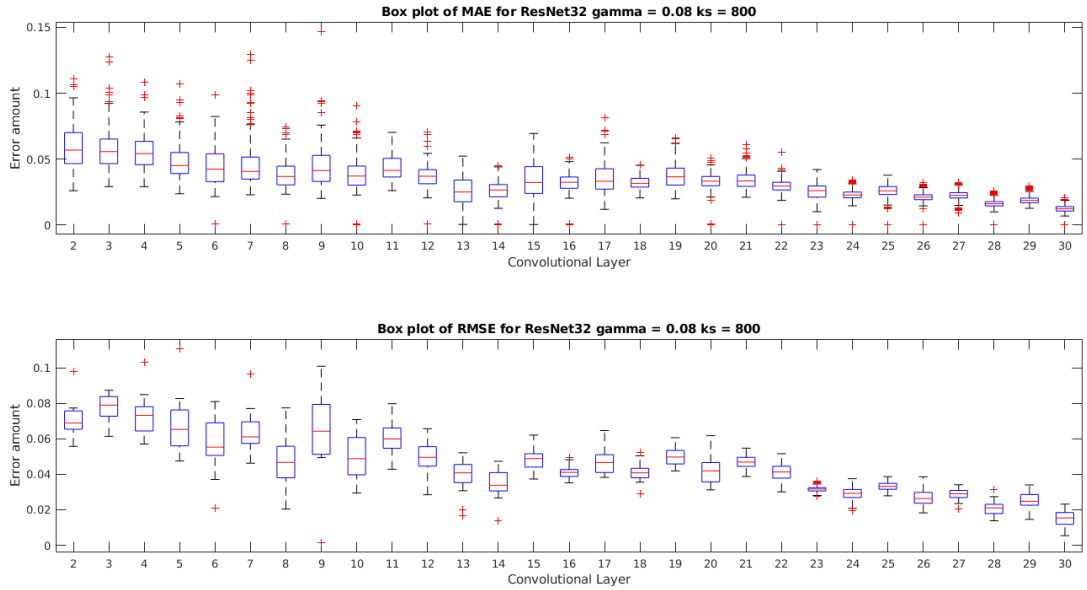

Figure 2.16: MAE and RMSE of predicted ResNet32 for $\gamma = 0.08$ and $ks = 800$

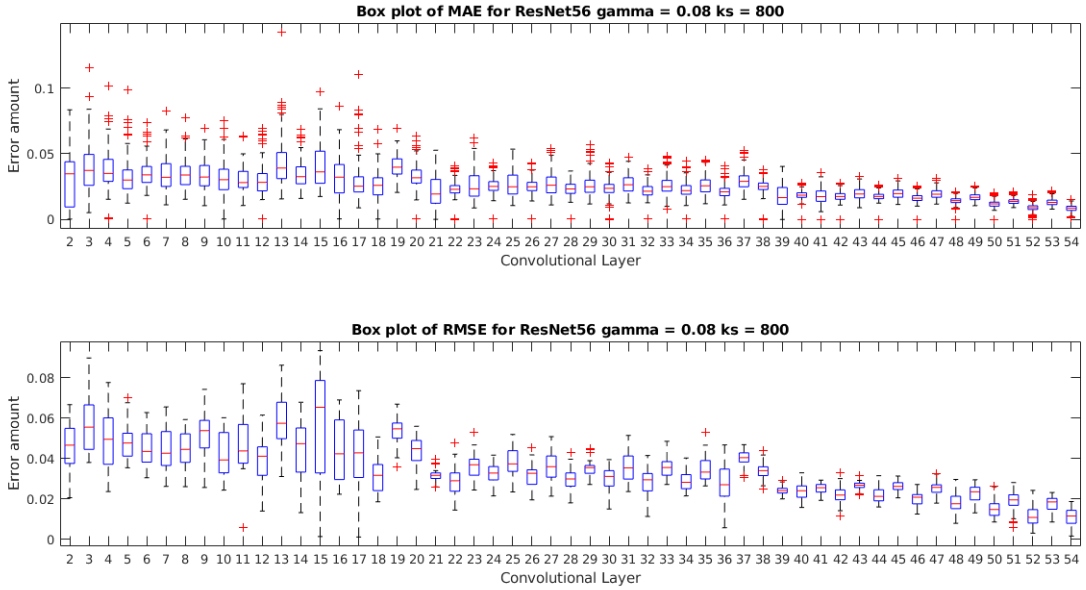

Figure 2.17: MAE and RMSE of predicted ResNet56 for $\gamma = 0.08$ and $ks = 800$

MAE and RMSE have their limitation: they only provide the amount of error, which is valuable if the operation of the weights involves only addition and subtraction. However, the cross-correlation operation consists of both multiplication and addition. The relative error measurements are useful in this scenario. So in order to better interpret the performance, I plot the percentage scaling(PS), as Equation 2.26.

$$PS_{it} = \frac{1}{J} \sum_{j=0}^{J} \frac{\bar{w}_{ijt}}{\hat{w}_{ijt}} \tag{2.26}$$

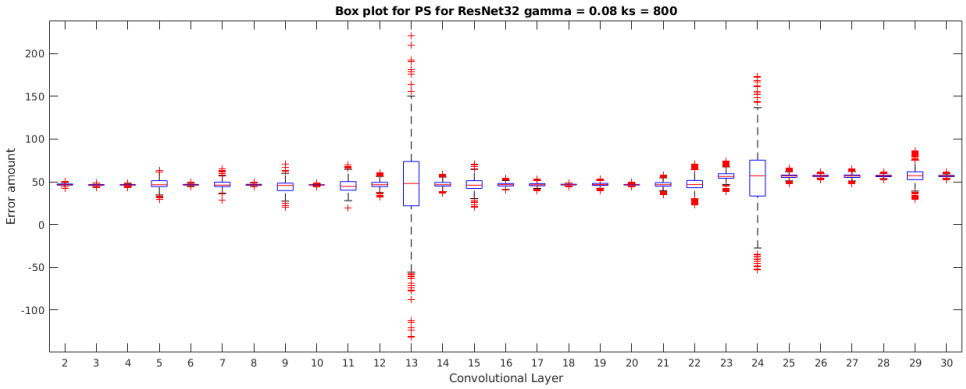

Figure 2.18: Percentage scaling(%) of predicted ResNet32 for $\gamma = 0.08$ and $ks = 800$

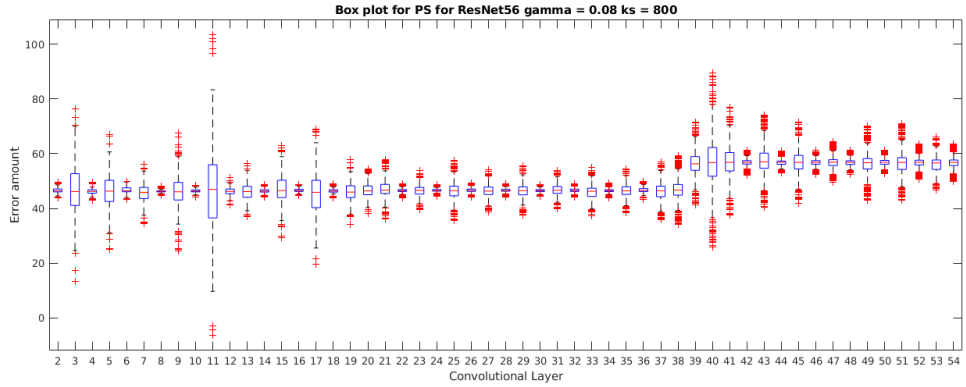

Figure 2.19: Percentage scaling(%) of predicted ResNet56 for $\gamma = 0.08$ and $ks = 800$

The Figure 2.18 to Figure 2.21 show the amount of scaling for parameters in the predicted ResNet32 and ResNet56 with $\gamma = 0.08$ and $\gamma = 1$. The plots suggest that the predicted value in most layers have a distribution that centers around 50%. Some of the distribution

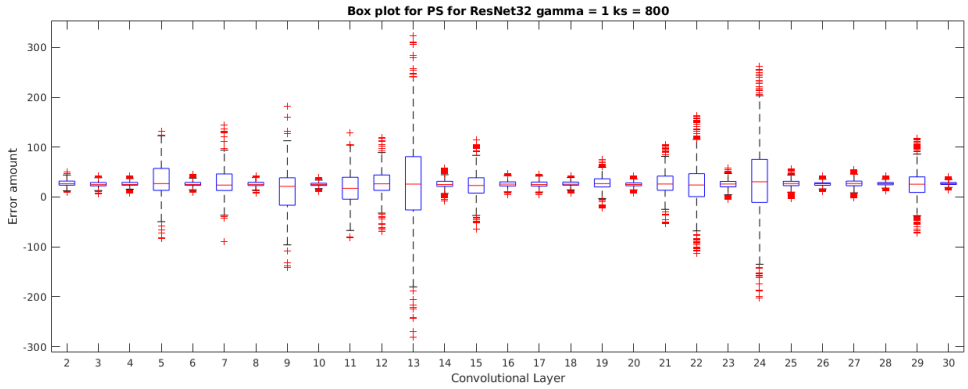

Figure 2.20: Percentage scaling(%) of predicted ResNet32 for $\gamma = 1$ and $ks = 800$

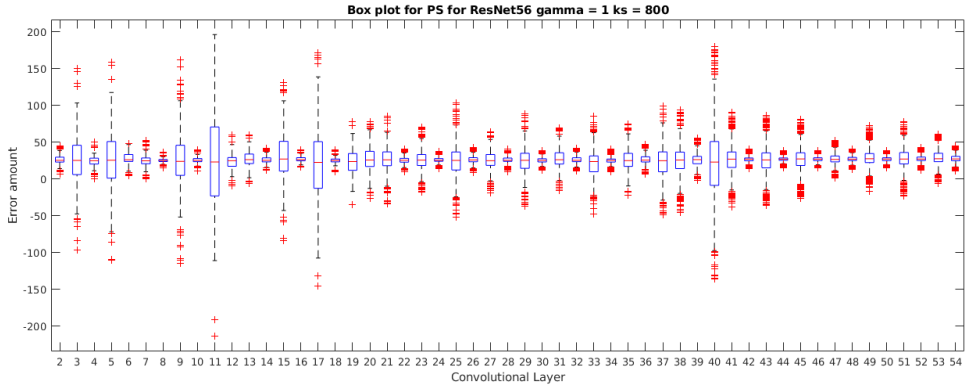

Figure 2.21: Percentage scaling(%) of predicted ResNet56 for $\gamma = 1$ and $ks = 800$

is narrow, which means that those convolutional filters are evenly scaled from the original filters. If all convolutional filters are evenly scaled, the feature map generated by the channel has the same property as the original feature map. However, in most of the odd layers, the distributions of the scaling have larger standard deviations. In these layers, the PSVMs are harder to generalize the relation between two layers. All the odd layers are the first layer in the ResNet building blocks, as shown in Figure 2.1. Inputs of these layers are the outputs of previous layers plus the residual from the previous blocks. This suggests that the weight distributions between layers are influenced by the additional short-cut path.

Based on all the experiments and results so far, the MLMs have the capability to rebuild the ResNets based only on the knowledge of the initial layer. The predicted ResNets can achieve

very similar performance to the pretrained ResNet even though there exists some difference in parameters.

2.4 Experiments: Noisy Predicted ResNet Models

At this stage, the predicted ResNet models can give similar predictions to the pretrained ResNet model. The next part of the research is to introduce noise to the MLM. I explore two possible places to add noises: Parameters in the initial layer and the inverse sigmoid function. For each place, I compare the performance of the predicted noisy model to the pretrained model, as well as the pretrained model with the same noise directly on parameters. In the following thesis, the "noisy predicted models" means the model produced by the MLM with either noise to the sampling or the initial layer, and the contrast models are called "noisy pretrained models," where the noises are added directly to the pretrained ResNets.

2.4.1 Noise at the Inverse Sigmoid Sampling

The first place to add noise is the inverse sigma function. When sampling from probabilities back to weights, I put an offset uniformly to all parameters, or a Gaussian noise separately to each parameter, as Equation 2.27. In the equation, w_{jpt} is a single weight parameter. It is sampled back from probability $p(w_{jpt}|w_{ij})$. n_s is the noise apply to the inverse sigmoid function. N_s are the set of numbers that follows the Gaussian distribution with μ and σ^2. For weights in each $p \in P$, there is an independent noisy set N_s. The Figure 2.22 shows the influence of ns on the inverse sigmoid function. The left plot is when $n_s = 0.1$, a fixed number, and the right plot is when n_s is a Gaussian random number where $\sigma = 0.1$.

$$\begin{gathered} \bar{w}_{jhk} = s^{-1}(p(\bar{w}_{jht}|w_{ij})) = \frac{1}{\beta}\left(log(p(\bar{w}_{jht}|w_{ij})) + log((1 - p(\bar{w}_{jht}|w_{ij})) + n_s\right) \\ n_s \in \mathbb{R} \; or \; n_s \in N_s, N_s \sim \mathcal{N}(\mu, \sigma^2) \end{gathered} \tag{2.27}$$

The control group is the pertained model that adds the same noise directly to each parameter, i.e., $\hat{w}_{jhk} + n_{sp}$ for every single weight parameters.

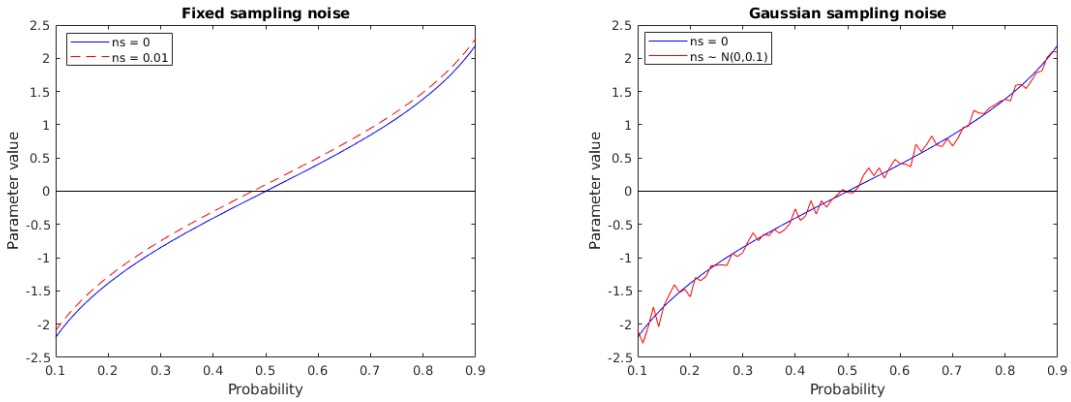

Figure 2.22: s^{-1} for fixed noise(left) and Gaussian noise(right)

2.4.2 Noise at the Initial Layer

The other place to add noises is the parameters in the initial layer, i.e., the layer that serves as the input to the MLM to predict the whole model. Similar to the sampling noise, the noise can be uniformly applied to all parameters or Gaussian distributed separately to each parameter.

$$\begin{aligned} \bar{w}_{ijt} &= \bar{w}_{ijt} + n_m \ , k = 0, 1, ..., 8 \\ n_m \in \mathbb{R} \ or \ n_m &\in N_m, N_m \sim \mathcal{N}(\mu, \sigma^2) \end{aligned} \tag{2.28}$$

The control group is the pretrained model with the same noise adding to the initial layer.

2.4.3 Experiment I: Fixed Noise

The first experiment is to test the influence of a fixed noise adding to either two places described in previous sections. Both the experimental group and the control group receive the same noise. All layers except the final fully connected classifier are frozen and then fine-tune the noisy models for five epochs. The learning rate is set to 0.0001 and the momentum to 0.9. Data is normalized as described in the original ResNet paper. The recorded results are the average of two runs. In the following thesis, the noise to the initial layers is denoted

by "NM," and the noise to the sampling is denoted by "NS." Table 2.3 shows the different noise value in this experiment.

NS	0.001	0.003	0.005	0.007	0.009	0.011	0.013	0.015
NS	0.0003	0.0005	0.0007	0.0009	0.0011	0.0013	0.0015	0.0017
NM	0.005	0.01	0.015	0.02	0.025	0.03	0.035	0.04

Table 2.3: Experiment setting: Fixed noise to initial layer and sampling process

Results. The experimental results for ResNet32 and ResNet56 are shown in Figure 2.23 to Figure 2.25. For both ResNet32 and ResNet56, the accuracy drops radically when NS noise is greater than 0.03, as suggested by Figure 2.23 and Figure 2.24. In contrast, with the same amount of NS noise, the accuracy drops more gently for pretrained ResNet32 and 56. For NS noise in the range 0.0003 to 0.0017, both pretrained and predicted ResNets have a fluctuant accuracy, but generally, NS noise within 0.0017 has a 0.2% impact on the accuracy. As NM, for both ResNet32 and ResNet56, with noise increasing from 0.005 to 0.04, the accuracy decreases approximately linearly. The amount of decrease is roughly 0.5% for ResNet32 and 0.4% for ResNet56.

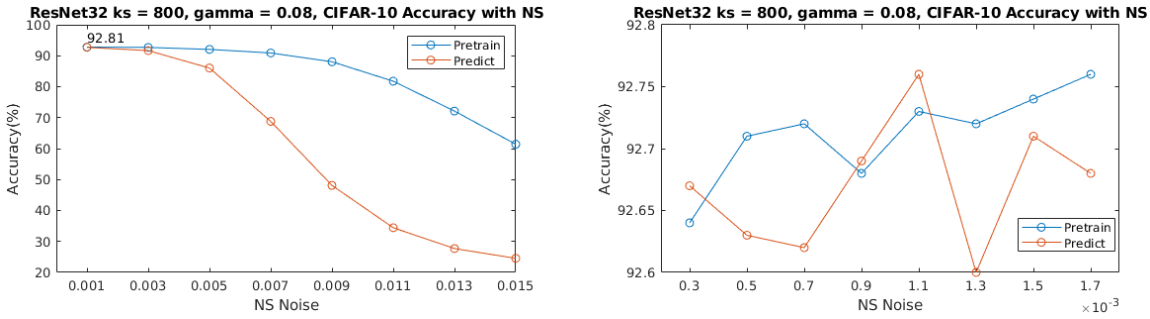

Figure 2.23: Performance of ResNet32 with fixed NS

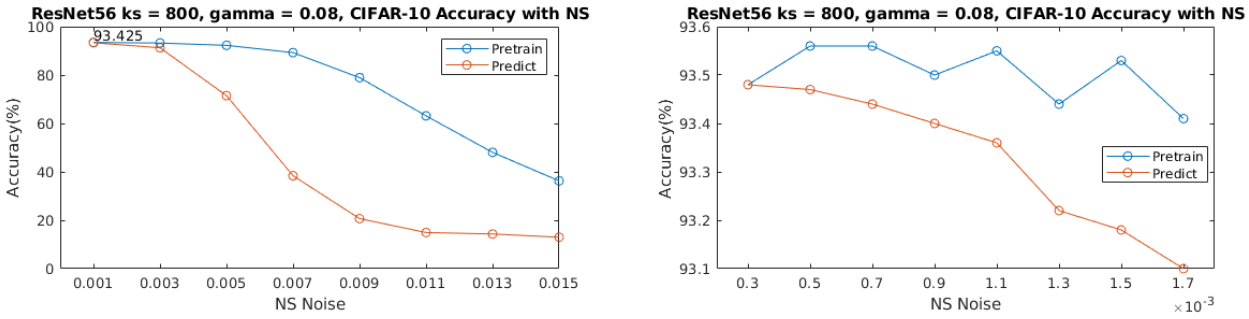

Figure 2.24: Performance of ResNet56 with fixed NS

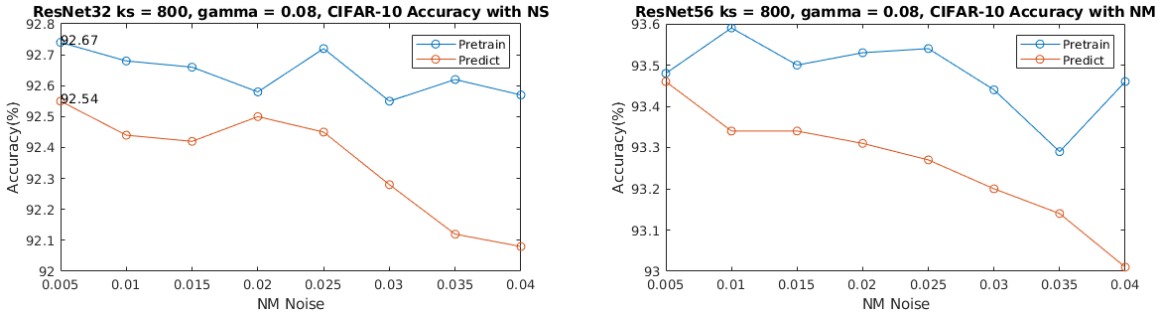

Figure 2.25: Performance of ResNet32 and ResNet56 with fixed NM

2.4.4 Experiment II: Gaussian Noise

The second experiment adds Gaussian noise to every parameters for both NM and NS. The experiment fixes the μ and tests the impact of different σ. Experimental settings are shown in Table 2.4. The training procedure is the same as it in the previous section.

	σ							
NS	0.001	0.003	0.005	0.007	0.009	0.011	0.013	0.015
NS	0.0003	0.0005	0.0007	0.0009	0.0011	0.0013	0.0015	0.0017
NM	0.005	0.01	0.015	0.02	0.025	0.03	0.035	0.04

Table 2.4: Experiment setting: Gaussian noisy with different σ to initial layer(NM) and sampling process(NS)

Results. The results are shown in Figure 2.26 to Figure 2.28. These plots suggest that the accuracy are pretty similar to the fixed noise model accuracy except that the pretrained ResNet with Gaussian noise to every parameter tends to have a much better performance than those with fixed noises, as the blue line in Figure 2.26 left and Figure 2.27 left.

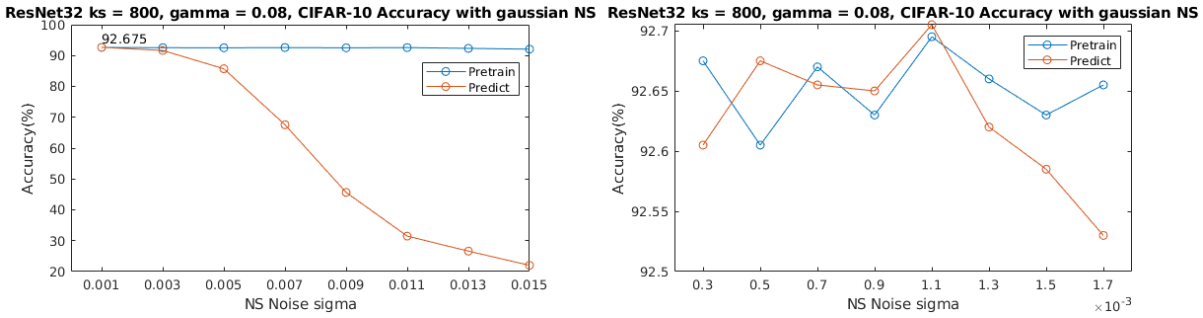

Figure 2.26: Performance of ResNet32 with Gaussian NS

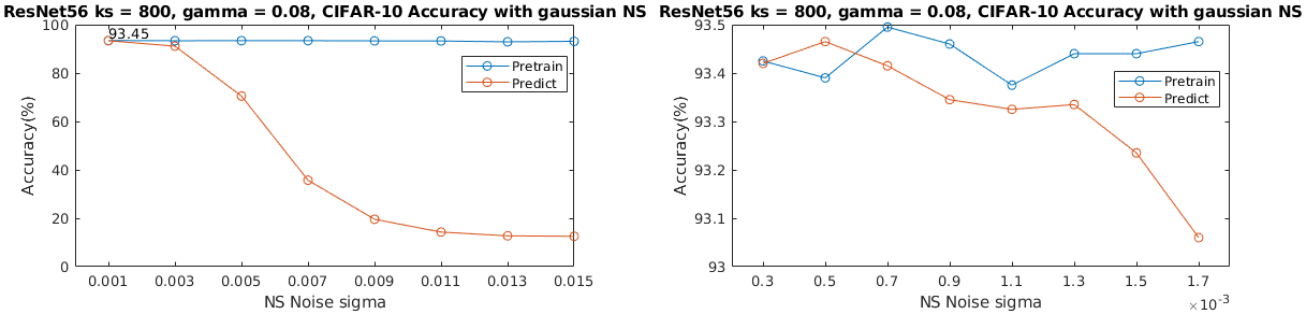

Figure 2.27: Performance of ResNet56 with Gaussian NS

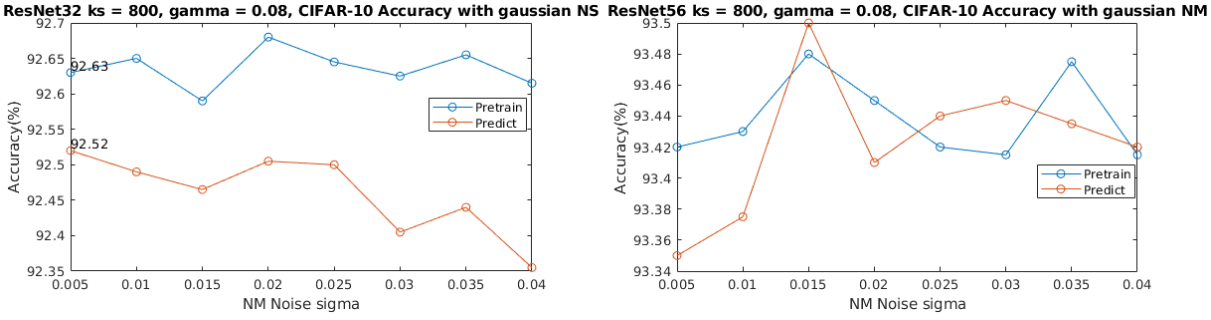

Figure 2.28: Performance of ResNet56 with Gaussian NM

2.5 Results

The experiment results without noise show that the MLM can generate models that have a performance very similar to the pretrained one, which suggests that the MLM is capable of generalizing the potential relation between layers, and this relation only dependents on the weights in the previous layer. However, the second experiment with noise shows that the model predicted by the MLM will diverge more severely to the original classification problem than pretrained models when a disturbance is added to the parameters. This partly contradicts the (ii) in my hypothesis where the relation should connect to the problem. This can be caused either by a lack of data in the shallow layers or overfitting in the deeper layer. However, it can be an advantage in the following experiments with ensemble learning. A model that slightly diverges from the original problem, if implement as a part of an ensemble, is possible to increase the generalization.

Chapter 3

Ensemble Models

With a small amount of noise either to the initial layer or to the inverse sigmoid function, the performance of the output models demonstrates that MLM can generate slightly different solutions to the original classification problem. Although most of the time, accuracies of the noisy predicted models are inferior to the original one, there is a possibility that they can be used to create a more generalized solution. The advantage of the noisy predicted models is that they have a very short training time comparing to the normal CNNs. Based on these, I explore the feasibility of creating an ensemble of noisy models that can give better performance but still maintain the advantages of training time.

3.1 Ensemble Methods

Unlike usual machine learning strategies that search for a single best function as the solution to a problem, ensemble learning generalizes the solution from a set of functions, or "weak learners." This set of functions are often called an ensemble. As suggested by Thomas Dietterich[5], each weak learner inside the ensemble should have a reasonably low error rate as well as some level of variance at the prediction. Generally, a weak learner for a neural network ensemble is trained in the normal way, i.e., iteratively going through forward and back-propagation over a dataset. Many ensemble learning methods like boosting and bagging[11] trained each individual weak learner simultaneous via negative correlate errors[12].

In the research, weak learners are noisy models created by the MLM. To accord with the advantages of training time for MLM generated noisy models, the ensemble models are

designed to exclude trainable parameters. In this thesis, I implement two weight-free fusion methods to generalize each weak learner's predictions: Averaging and Plurality voting.

3.1.1 Plurality Voting

For one input data, every weak learner votes for the label that has the highest output probability, see Figure 3.1. The final decision is the label that has the highest vote. In the figure, all arrows are parameter-free, which means that the method requires no further training.

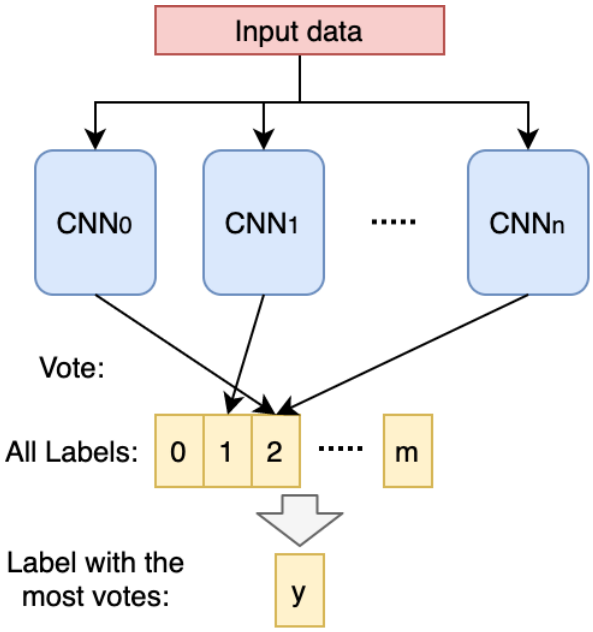

Figure 3.1: Structure of ensemble of CNNs with plurality vote

3.1.2 Averaging

For a single input data, every weak learner generates their own probability prediction for each label, and the final decision is made based on the average of all predictions, see Figure 3.2. The method can be parameter-free if all weights w are equal to 1.

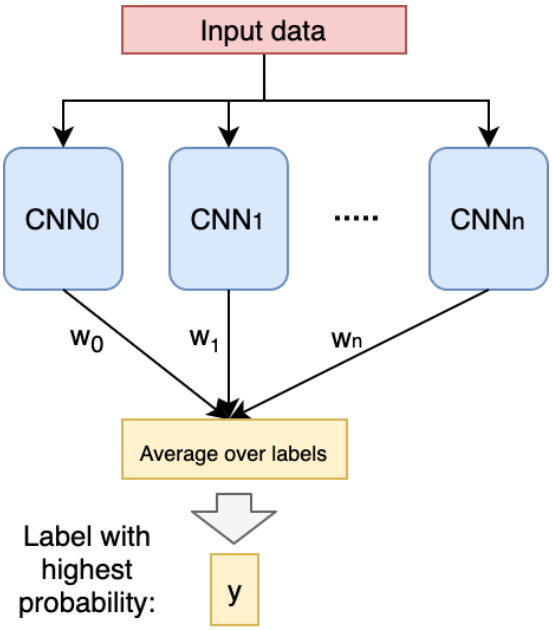

Figure 3.2: Structure of ensemble of CNNs with Averaging

3.2 Experiments and Results

To discover the effectiveness of ensemble models, I establish the experiments on noisy models generated by both methods described in Chapter 2. The models are noisy models based on ResNet32, ResNet56[7] that are pretrained with CIFAR-10 dataset. Since generating noisy CNNs requires fine-tuning the final fully-connected layers, all pretrained models in this experiment are fine-tuned. The model performance is shown on Table 3.1.

Method	Acc@1(Paper)	Acc@1(This exp.)	params
ResNet32	92.49	92.63	0.46M
ResNet56	93.03	93.52	0.85M

Table 3.1: Performance(%) of baseline model implemented to generate the noisy model

3.2.1 Experimental Setups

Noisy models that with initial layer noise and sampling noise are named "NM" and "NS." The noise is either constant noise or Gaussian noise with μ and σ^2. r is the offset to every parameter in the initial layer for NM models or every parameter for NS models. All models are created based on fine-tuned pretrained models. All noisy models generated by the MLM will be fine-tuned on the last fully-connected layer for five epochs with a learning rate of 0.0001 over the full CIFAR-10 dataset. This is also the setting implemented to fine-tune the pretrained models. The control group of all experiments is the ensemble model construct based on noisy pretrained models described in Chapter 2. Every single noisy pretrained model will also be fine-tuned with the above setting.

3.2.2 Experiments I: Ensemble of ResNet32 and ResNet56

The first experiments are on the ResNet32 and ResNet56 for both NM models and NS models. For each type, I define experimental groups with different size, μ, σ, and use MLM to generate corresponding models. All noisy predicted and pretrined models in a group form four ensemble models with averaging and plurality vote methods. The ensemble models are then evaluated on the CIFAR-10 test set. The experimental settings are shown in Table 3.2.

NS(μ)	0	0	0	0
NM(μ)	0	0	0	0
NS(σ)	0.001	0.0015	0.002	0.0025
NM(σ)	0.005	0.015	0.025	0.035
Method	Averaging	Plurality Vote		
Model per ensemble	8			
CNNs	ResNet32	ResNet56		
Model training	5 epochs after adding noise			
Training layers	Only fully connected layer			

Table 3.2: Experiment settings for ResNet Noisy ensemble

As the table shows, the experimental groups are ResNet32 and ResNet56 predicted by the MLM with Gaussian noise to both NM and NS. Each ensemble is formed with eight models

with the same μ and σ. The control group have the same settings except that the CNN is the pretrained noisy as described in Chapter 2.

Results. The results of the ensemble models on the CIFAR-10 test-set are shown in Figure 3.3 and Figure 3.4. The performance of both the experimental group and control groups did not meet the expectation. Generally, pretrained ensemble groups have better accuracy than the predicted ensemble groups, but both ensemble groups are under or have no obvious advantage over a single pretrained ResNet. The highest performance has only 0.2% accuracy over the baseline model (ResNet32 NM pretrained averaging). This means that all the models in an ensemble tend to have very close classifications, which is not expected for weak learners. As described by Thomas Dietterich[5], each individual model in an ensemble should have some variance from each other.

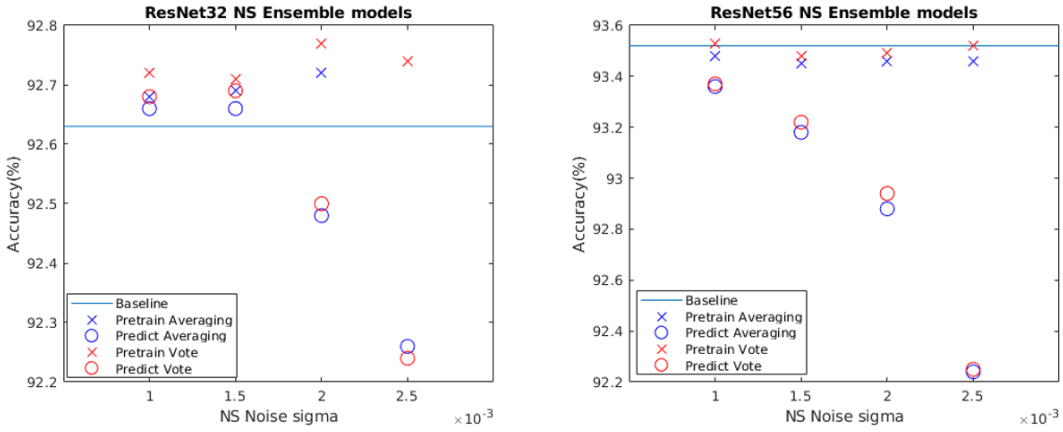

Figure 3.3: Accuracy for ensemble models with NS noise for experiment I

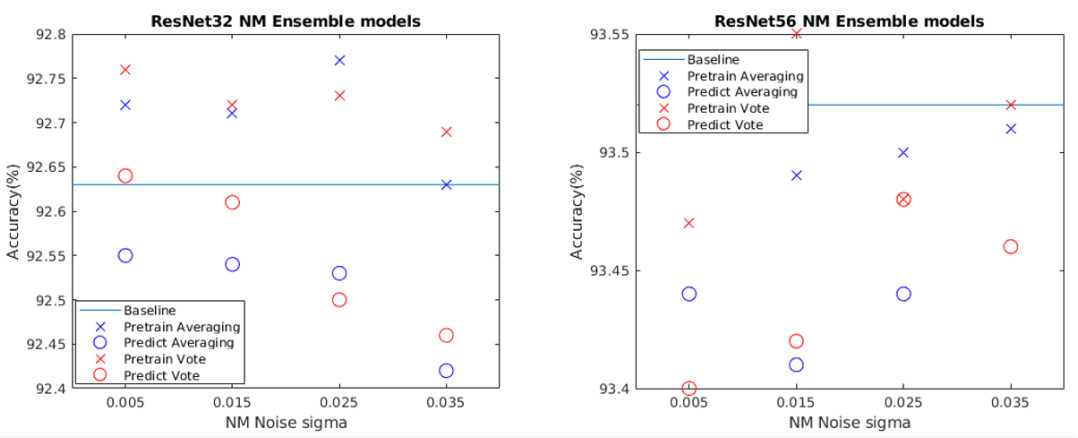

Figure 3.4: Accuracy for ensemble models with NM noise for the experiment I

3.2.3 Experiment II: Rethink the Training Layers

Based on previous results, in this experiment, all models will be fine-tuned for both the last group of convolutional layers and the final fully connected layer, and the rest of the settings remain the same, as shown in Table 3.3. The last group of convolutional layers for both ResNet32 and ResNet56 have a dimension of 64, named as "CL64". For ResNet32, CL64 are convolutional layer 20 to convolutional layer 30, and for ResNet56, CL64 are convolutional layer 36 to 54.

NS(μ)	0	0	0	0
NM(μ)	0	0	0	0
NS(σ)	0.001	0.0015	0.002	0.0025
NM(σ)	0.005	0.015	0.025	0.035
Methods	Averaging	Plurality Vote		
Model per ensemble	8			
CNNs	ResNet32	ResNet56		
Model training	5 epochs after adding noise			
Training layers	Fully connected layer	CL 64		

Table 3.3: Experiment settings for ResNet Noisy ensemble

Results. The results of this experiment are shown in Figure 3.5 to Figure 3.6. From the plots, we can see that both pretrained noisy models and predicted noisy models have better performance than the baseline model. NM noisy predicted models have the highest accuracy for ResNet56, and NS noisy predicted model has the best accuracy for ResNet32. Also, the plots show that the noisy predicted models are more fluctuate than the noisy pertained model, and this fluctuation can lead to a better generalization of an ensemble. So, for ResNet32 and ResNet56, the MLM with noise generates models with more variation. By retraining the deeper layers for only 5 epochs, the new models could give a better generalization by forming an ensemble. So this method increases the accuracy with a very short training time.

In addition to the experiments shown in the plots, I also did many experiments with different noise values and ensemble settings. The best ensemble model I got in this thesis is shown in Table 3.4.

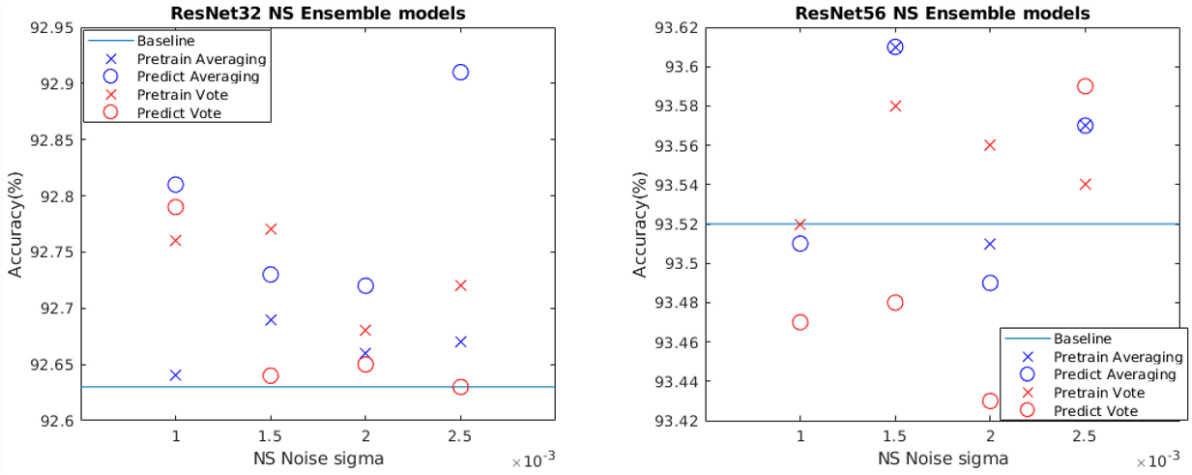

Figure 3.5: Accuracy for ensemble models with NS noise with training the CL 64 and fully connected layer.

Model	Noise Type	μ, σ	Ensemble method	Model per ensemble	Accuracy
ResNet32	NM Gaussian	$0, 0005$	Average	8	93.13%
ResNet56	NM Gaussian	$0, 0.035$	Vote	8	93.79%

Table 3.4: Best performance with this experiment so far

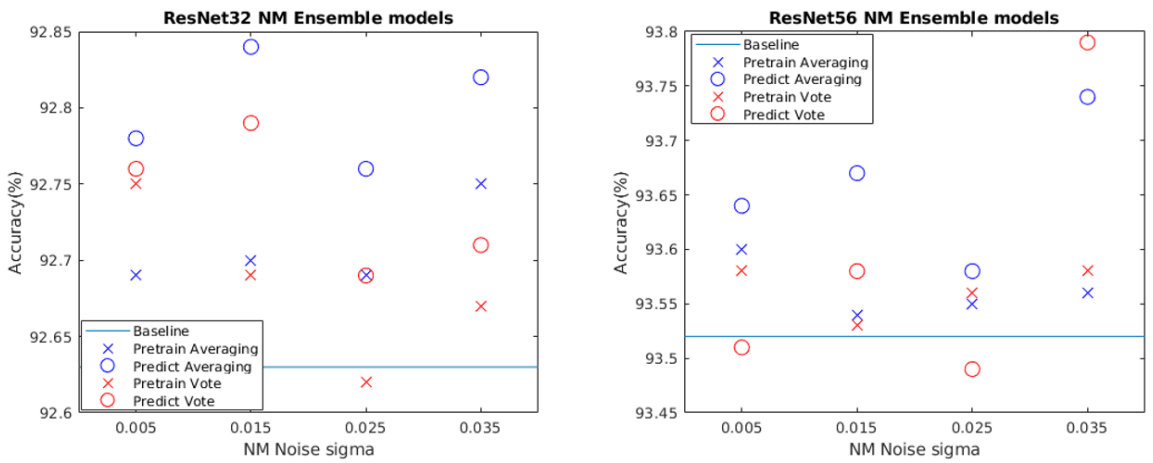

Figure 3.6: Accuracy for ensemble models with NS noise with training the CL 64 and fully connected layer.

Chapter 4

Conclusion and Future works

4.1 Conlusion

This thesis studies the relation of parameters in convolutional neural networks and its application. The thesis begins with a hypothesis about internal patterns of parameters and proposes a machine learning model to study this relation. The experimental results suggest that the MLM based on this hypothesis successfully rebuilds the neural network model that can give a similar prediction. In addition to this, the thesis uses experiments to demonstrate the influence of noise to the MLM. Using the noisy model, the thesis explores a possible way to use ensemble methods to increase the performance to the original problem. The relation between parameters have a great potential in studying novel way to adjust neural networks. This thesis contributes a direction for future works in modeling weight distributions and studying non-linearity of neural networks.

4.2 Future Works

There is a great potential to use the MLM to compress neural network models. In this experiment, in order to generate results that close to the original problems for ensemble models, the training set for the MLM is large. However, near the end of this research, I also studied the possibility to use only part of the data as training set. Due to time limitation, I only implemented a few general experiments, and the best results are shown in Table 4.1. The idea is to reduce the size of training sets in the final groups of convolutional layers and then retrain the network. Future study can be done to further increase the amount of compression by carefully selecting parameters that best interpret the relation.

Model	Original Acc.	Compressed Acc	Params reduction	method
ResNet32	92.63%	88.31%	49%	CL 64 30% training data
ResNet56	93.52%	89.61%	49%	CL 64 30% training data

Table 4.1: Simple test results for parameter compression

4.3 Reference

[1] Buhrmester V, Münch D, Arens M., Analysis of explainers of black box deep neural networks for computer vision: A survey. arXiv:1911.12116. 2019 Nov 27.

[2] Boser, B.E., Guyon, I.M. Vapnik, V.N. A training algorithm for optimal margin classifiers, *In Proceedings of the Fifth Annual Workshop of Computational Learning Theory*, Pittsburgh

[3] Platt, John et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods, *Advances in large margin classifiers*, 61–74, 1999.

[4] S. Chakrabartty and G. Cauwenberghs, Gini-support vector machine: Quadratic entropy based multi-class probability regression, *J. M. L. R.*, vol. 8, pp. 813–839, Apr. 2007.

[5] T. G. Dietterich, Ensemble learning, *The Handbook of Brain Theory and Neural Networks*, M. A. Arbib, Ed. (MIT Press, ed. 2, 2002), pp. 405–408.

[6] Y. Abouelnaga, O. S. Ali, H. Rady, and M. Moustafa, Cifar-10: KNN based ensemble of classifiers, *Inter. Conf. on Computa. Sci. Computa. Intelli. (CSCI)*, pp. 1192–1195, Dec 2016.

[7] He, K., Zhang, X., Ren, S., Sun, J., Deep residual learning for image recognition, arXiv:1512.03385, 2015.

[8] Gastaldi, Xavier, Shake-shake regularization of 3-branch residual networks, *In ICLR Workshop Track*, 2016.

[9] Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images, *Report*, 2009.

[10] Huang, Z, Wang, X, and Luo, P, Convolution-Weight-Distribution Assumption: Rethinking the Criteria of Channel Pruning. arXiv:2004.11627.

[11] M. M. Islam, X. Yao, S. M. S. Nirjon, M.A. Islam, and K. Murase, Bagging and Boosting Negatively Correlated Neural Networks, *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, Vol.38(3), pp. 771-784, 2008.

[12] Liu, Y. and Yao, X. Ensemble learning via negative correlation. *Neural Networks*, 12(10):1399–1404, 1999a.

[13] Chenglong, Z., Bingbing, N., Jian, Z., Qiwei, Z., Wenjun, Z., and Qi, T., Variational convolutional neural network pruning, *CVPR*, 2019. 02

[14] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, Importance estimation for neural network pruning. In *CVPR*, 2019

[15] He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., and Han, S. Amc: Automl for model compression and acceleration on mobile devices. *ECCV*, 2018.
[16] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *ICLR*, 2017.
[17] Liu, C., Zoph, B., Shlens, J., Hua, W., Li, L., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K.. Progressive neural architecture search. arxiv:1712.00559, 2017.
[18] Pham, H. , M. Y. Guan, Zoph, B. , Q. V. Le, and Dean, J. .Efficient neural architecture search via parameter sharing. *ICML*, 2018.
[19] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. MnasNet: Platform-aware neural architecture search for mobile. *CVPR*, 2019.